

الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université Akli Mohand Oulhadj - Bouira -

X•⊙V•εX •κ||ε □:κ:|∧ :||κ•X - X:⊙ε⊙÷t -



وزارة التعليم العالي والبحث العلمي
جامعة أكلي محمد أوحاج
- البويرة -

Faculté des Sciences et des Sciences Appliquées

كلية العلوم والعلوم التطبيقية

Département de Génie électrique

Polycopié de cours

En: Télécommunications, Électronique

Spécialité: Systèmes des Télécommunication, Électronique et
Système Embarqué

Niveau : Master



Linux System

Dr. Bilal SAOUD

Année: 2022-2023

Introduction	4
Chapter 1: Operating system introduction	6
1.1 Introduction	6
1.2 Computer architecture	6
1.3 Definition of OS	7
1.4 History of Linux	6
1.5 Free software	9
1.6 Linux distributions	11
1.7 Conclusion	13
1.8 Exercises	13
Chapter 2: The Bash shell	14
2.1 Introduction	14
2.2 Shell's types	15
2.3 Entering Linux commands	15
2.4 Command man	16
2.5 Linux commands	17
2.6 Bash features	19
2.7 Bash interpreter	20
2.8 Conclusion	21
2.9 Exercises	21
Chapter 3: Linux applications	23
3.1 Introduction	23
3.2 Text editors	23
3.3 Productivity software	27
3.4 Latex	28
3.5 Encryption software	31
3.6 Network software	32
3.7 Conclusion	33
3.8 Exercises	34
Chapter 4: Shell scripting	35
4.1 Introduction	35
4.2 Simple scripting	35
4.3 Variables, assignments and parameters	35
4.4 Input and output	36
4.5 Statements	37
4.6 Loops	39
4.7 Arrays	40

4.8 Functions	41
4.9 Conclusion	42
4.10 Exercises	42
References	43

Learning the Linux operating system offers several benefits and opportunities for individuals in various fields. Today, Linux becomes very popular. It is used in all type of computers and its used is spreading day after day. Furthermore, Linux is also very popular as servers' operating systems in computer networks. Today many developers prefer to develop software for Linux. It is a mandatory to learn some basic fundamental about Linux. Here are some compelling reasons why learning Linux is valuable:

1. **Widely used and versatile:** Linux powers a significant portion of the world's servers, supercomputers, embedded systems, and networking devices. It is the foundation of many popular technologies, including Android, cloud computing platforms, and IoT devices. By learning Linux, you gain skills that are highly relevant in today's technology-driven world.
2. **Career Opportunities:** Linux skills are in high demand in the job market. Many organizations, particularly in the IT industry, seek professionals with Linux expertise for roles such as system administrators, network administrators, DevOps engineers, cybersecurity specialists, cloud engineers, and software developers. Having Linux knowledge can open up a range of career opportunities and increase your marketability.
3. **Cost-effective solution:** Linux is known for its open-source nature, meaning it is available for free and can be modified and distributed without restrictions. This makes Linux an attractive option for organizations looking to minimize software licensing costs. By understanding Linux, you can contribute to cost-effective solutions and help organizations save money.
4. **Flexibility and customizability:** Linux offers a high degree of flexibility and customization. Its open-source nature allows users to modify and tailor the operating system to their specific needs. Learning Linux empowers you to personalize your computing environment, customize software packages, and fine-tune system performance according to your requirements.
5. **Security and stability:** Linux is renowned for its security and stability. Its architecture and design principles prioritize security, and the open-source community actively collaborates to identify and patch vulnerabilities. By learning Linux, you gain insight into robust security practices and can contribute to creating secure computing environments.
6. **Learning and problem-solving:** Linux provides a rich learning environment. As you delve into Linux, you encounter command-line interfaces, shell scripting, system configuration, and troubleshooting. Mastering these aspects enhances your problem-solving skills, boosts your understanding of operating systems, and develops your overall technical acumen.
7. **Collaboration and community:** The Linux community is vast and vibrant. Engaging with the Linux community exposes you to a diverse range of knowledge, expertise, and support. You can join forums, mailing lists, and open-source projects, collaborate with like-minded individuals, and contribute to the ongoing development of Linux-based software.

8. Transferable skills: Linux knowledge goes beyond the operating system itself. It helps you grasp fundamental concepts of computer systems, networking, security, and software development. The skills you acquire while learning Linux can be applied to other operating systems and technologies, making you adaptable and capable of quickly learning new tools and platforms.
9. Personal use and empowerment: Learning Linux extends beyond professional applications. It allows you to harness the power of a robust, customizable, and secure operating system for your personal computing needs. Whether you want to set up a home server, experiment with new software, or explore the inner workings of technology, Linux provides a platform for exploration and personal growth.

Finally, learning Linux equips you with valuable skills, enhances your career prospects, and empowers you to contribute to a wide range of technological domains. Whether you are pursuing a career in Electronics, Telecommunication, IT or simply seeking to expand your knowledge, Linux offers a rewarding and enriching learning journey.

Chapter 1: Operating system introduction

1.1 Introduction

Operating system (OS) is an important part in computer or any electronic devices. It is crucial part that allows hardware architecture to work. For instance, when we say a node in a computer network topology it is clear that we refer to a device that has the two fundamentals parts, which are software (OS) and hardware. In addition, a computer in network refers also to software and hardware. In this chapter, some fundamental concepts about OS will be presented in order to understand the functionality of OS. First of all, computer architecture will be presented. Then, the definition of OS is given. After that, we will present Linux OS.

1.2 Computer architecture

Computer architecture refers to the structure and organization of computer systems, including the design principles, components, and interconnections that enable the execution of instructions and data processing. It encompasses both the hardware and software aspects of a computer system. The following figure illustrates the adopted architecture of computer.

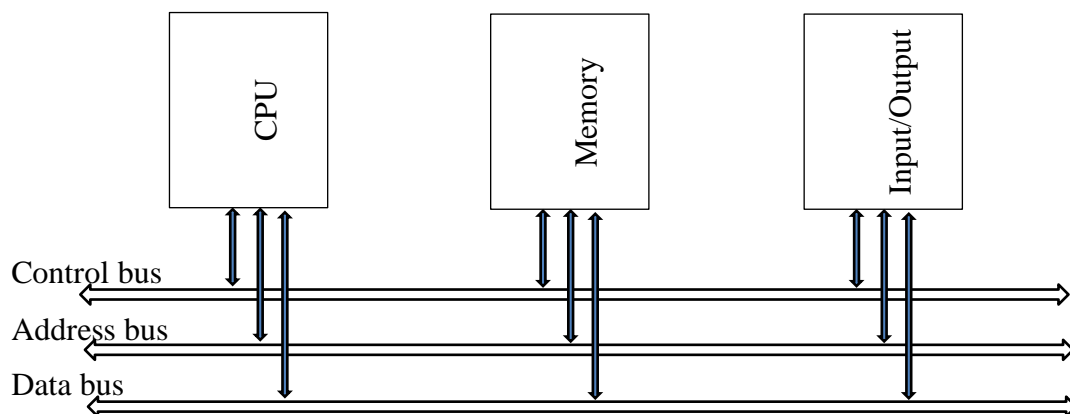


Figure 1: Architecture of computer.

Here are key components and concepts related to computer architecture:

- **Central Processing Unit (CPU):** The CPU is the primary component responsible for executing instructions and performing calculations. It consists of the arithmetic logic unit (ALU) for mathematical operations, control unit for instruction interpretation and sequencing, and registers for temporary data storage.
- **Memory:** Computer systems have different types of memory for storing data and instructions. This includes:

- Random Access Memory (RAM): RAM provides temporary storage for data and instructions that the CPU can quickly access. It is volatile, meaning its contents are lost when power is turned off.
- Read-Only Memory (ROM): ROM contains permanent instructions or data that cannot be modified. It is used for firmware and system initialization.
- Cache Memory: Cache memory is a small, high-speed memory located closer to the CPU. It stores frequently accessed data to reduce memory latency and speed up execution.
- Bus: A bus is a communication pathway that enables data transfer between different components of a computer system. It includes the address bus for specifying memory locations, the data bus for transferring data, and the control bus for coordinating and synchronizing operations.
- Instruction Set Architecture (ISA): ISA defines the set of instructions that a CPU can execute and the programming model available to software developers. It includes instruction formats, addressing modes, and registers accessible to programs.
- Input/Output (I/O): I/O devices allow communication between the computer system and external devices. This includes devices such as keyboards, mice, displays, printers, storage devices, and network interfaces.
- Pipelining: Pipelining is a technique used to increase CPU efficiency by overlapping instruction execution stages. It divides the instruction execution into stages, allowing multiple instructions to be processed simultaneously.
- Parallel Processing: Parallel processing involves using multiple processors or cores to execute tasks simultaneously, thereby increasing overall computational power and performance.
- Instruction Level Parallelism (ILP): ILP refers to the ability of a processor to execute multiple instructions in parallel within a single program. Techniques such as superscalar execution and out-of-order execution are used to exploit ILP.
- System-on-Chip (SoC): SoC refers to integrating multiple components, including CPU, memory, I/O interfaces, and other system components, onto a single chip. It is commonly used in embedded systems and mobile devices.

We mention those computers are developed based on Von Neumann architecture, named after mathematician and computer scientist John von Neumann. It is based on a single shared memory space for both instructions and data (see figure 1). However, there are existed other architectures. For instance, we can cite the Harvard architecture. It has separate memory spaces for instructions and data, allowing simultaneous access to both.

Computer architecture plays a fundamental role in determining the performance, efficiency, and capabilities of computer systems. Architects and designers strive to balance factors such as instruction execution speed, memory capacity and latency, power consumption, and cost to create optimized and scalable computer systems.

1.3 Definition of OS

An operating system (OS) is a software program that serves as the foundation for managing computer hardware and software resources, enabling the execution of other

programs and providing a user-friendly interface. It acts as an intermediary between the user, applications, and the underlying hardware components of a computer system.

OS provides many functionalities to users. In addition, the OS will be the intermediate layer between the user and the computer hardware. The important part of OS is the Kernel. Here are some functionalities of an OS:

- Kernel: The core component of an operating system that directly interacts with the hardware and manages system resources such as memory, processes, and input/output (I/O) devices.
- Process management: The OS manages multiple processes, which are instances of running programs, allocating system resources, scheduling their execution, and facilitating inter-process communication.
- Memory management: Operating systems allocate and manage system memory resources, ensuring efficient utilization and providing processes with the necessary memory space for execution.
- File system: An operating system provides a file system that organizes and manages data on storage devices such as hard drives or solid-state drives. It handles file creation, deletion, organization, and access permissions.
- Device drivers: These are software components that allow the operating system to communicate with specific hardware devices, enabling the system to control and utilize devices like printers, keyboards, graphics cards, and network adapters.
- User interface: Operating systems provide a user interface (UI) that allows users to interact with the computer system. This can include graphical user interfaces (GUIs) with windows, icons, and menus or command-line interfaces (CLIs) that rely on text-based commands.
- Multiuser/Multitasking: Many modern operating systems support multiple users simultaneously and provide multitasking capabilities, allowing multiple programs to run concurrently on the same system.
- Security: Operating systems incorporate various security measures to protect the system and its data. This can include user authentication, access control mechanisms, encryption, and protection against malicious software.
- Virtualization: Some operating systems support virtualization, which allows multiple operating systems or instances to run simultaneously on a single physical machine, providing isolation and resource allocation between them.

We can find many examples of OS. Some of them are designed for computers and others for embedding systems. Among the popular OS examples, we can find:

- Microsoft Windows,
- macOS,
- Linux,

- Unix
- Android,
- iOS,
- etc.

Finally, an OS acts as a crucial intermediary layer between hardware and software, providing an environment that enables efficient and secure utilization of computer resources and allows users to interact with the system effectively.

1.4 History of Linux

The Linux OS, born out of the passion for freedom, collaboration, and open-source principles, has emerged as a powerful force in the world of technology. In this section, we will give history about Linux OS, tracing its origins, development, and the profound impact it has had on the computing landscape. It is worth to mention that Linux had been developed after many years of designing and enhancements. Many persons had collaborated at the creation of this OS.

Linux is part of the class of UNIX systems, developed and used since the 70s. UNIX is also an OS that has left an indelible mark on the world of computing, has a rich and fascinating history. Developed by a group of talented individuals at Bell Labs in the late 1960s, UNIX revolutionized the way computers were used, setting the stage for the modern computing landscape. The roots of UNIX can be traced back to the Multics project, an ambitious operating system initiative jointly undertaken by Bell Labs, MIT, and General Electric. When Bell Labs withdrew from the Multics project, Ken Thompson, Dennis Ritchie, and other researchers seized the opportunity to develop a simpler and more practical operating system. In 1969, they created the first version of UNIX, initially running on a Digital Equipment Corporation (DEC) PDP-7 minicomputer. As UNIX gained popularity within Bell Labs, refinements and enhancements were made, leading to the creation of the UNIX Time-Sharing System (UNIX TS) in 1971. UNIX was built on a set of guiding principles known as the "Unix Philosophy," emphasizing simplicity, modular design, and the power of small, focused tools. These principles would prove instrumental in UNIX's future success and widespread adoption.

In the early 1970s, UNIX began to make its way outside Bell Labs. Universities and research institutions embraced UNIX due to its portability, flexibility, and ability to run on various hardware architectures. UNIX's availability on different platforms allowed for collaboration and the sharing of software and ideas. As a result, different versions of UNIX, such as BSD (Berkeley Software Distribution), System V, and Xenix, emerged, each with its own features and characteristics. One of the notable branches of UNIX was BSD, developed at the University of California, Berkeley. BSD UNIX introduced significant enhancements, including the TCP/IP networking stack, which played a crucial role in the development of the internet. BSD also focused on open-source development, contributing to the spirit of collaboration and sharing that would shape the future of UNIX. In the 1980s, UNIX entered the commercial realm as various companies developed their own UNIX-based operating systems. AT&T's System V UNIX and the BSD variants were two of the most prominent branches during this period. To ensure compatibility and interoperability, efforts were made to standardize UNIX. The Open Group's Single UNIX Specification (SUS) emerged as a

standard that defined the core UNIX functionality and APIs, ensuring a level of portability across different UNIX flavors.

In the early 1990s, Linus Torvalds, a Finnish computer science student, embarked on a personal project to create a Unix-like operating system kernel. His intention was to build a system that would be free, flexible, and accessible to anyone interested in its development. Torvalds released the initial version of the Linux kernel on September 17, 1991, which laid the foundation for what would become a global phenomenon. Furthermore, central to the Linux story is the open-source philosophy that underpins its development. By making the source code freely available, Torvalds fostered a culture of collaboration and transparency, attracting a community of programmers who eagerly contributed their expertise to enhance and refine the operating system. This collaborative effort allowed Linux to evolve rapidly, gaining robustness and stability with each iteration.

Linux's growth accelerated in the late 1990s and early 2000s, as major corporations and organizations recognized its potential. Companies such as IBM, Red Hat, and SUSE invested in Linux, providing resources, support, and services that propelled its adoption in enterprise environments. Linux diversified its presence beyond traditional servers and found its way into embedded systems, supercomputers, smartphones, and other devices, demonstrating its versatility and adaptability. In addition, Linux's influence on the Internet cannot be overstated. The robustness, reliability, and security of the operating system made it the preferred choice for powering web servers, which played a vital role in the rapid growth of the World Wide Web. The LAMP stack (Linux, Apache, MySQL, and PHP) became a dominant force in web development, fostering innovation and transforming the digital landscape.

From its humble beginnings as a personal project, Linux has grown into a global phenomenon that powers critical infrastructure, drives innovation, and exemplifies the potential of collaborative development. The Linux community's commitment to openness and freedom has fundamentally reshaped the way we think about software and its distribution. As we move forward, Linux's legacy continues to inspire and shape the future of technology, promising a world where knowledge is shared, creativity thrives, and technological progress knows no bounds.

1.5 Free software

Linux has been developed based on the idea of free software. Free software refers to computer software that is distributed with certain freedoms that allow users to use, study, modify, and distribute it without restrictions. The concept of free software focuses on the idea of user freedom rather than price. Some key aspects and principles related to free software are given below:

- The Four Freedoms: Free software adheres to the four essential freedoms as defined by the Free Software Foundation (FSF). These freedoms include the freedom to:
 - Run the software for any purpose.
 - Study and modify the software's source code.
 - Redistribute exact copies of the software.
 - Distribute modified versions of the software.
- Open Source vs. Free Software: While the terms "free software" and "open source software" are often used interchangeably, they have slightly different focuses. Free software emphasizes the freedom aspect, whereas open source software focuses

more on the availability of the source code. However, both movements share similar principles and values regarding user freedom and collaboration.

- **Copyleft:** Copyleft is a licensing approach commonly associated with free software. It ensures that the freedoms granted to users are perpetuated even when modifications or derivatives of the software are made. Copyleft licenses, such as the GNU General Public License (GPL), require that any modified or derived works also be distributed as free software, ensuring the continued availability of user freedoms.
- **Community and Collaboration:** Free software thrives on community participation and collaboration. Developers and users come together to contribute to the improvement, testing, and bug-fixing of the software. This collective effort fosters innovation, knowledge sharing, and the creation of high-quality software.

Free software offers several benefits and advantages, including:

- **User Freedom:** Free software ensures that users have control over the software they use, allowing them to adapt it to their needs and preferences.
- **Security and Reliability:** With the source code available for scrutiny, security vulnerabilities can be identified and fixed promptly. Additionally, the collaborative nature of free software development often results in more stable and reliable software.
- **Cost:** Free software is typically available at no cost, eliminating the need for licensing fees. This makes it accessible to a wide range of users, including individuals, educational institutions, and organizations.
- **Customizability and Flexibility:** Free software can be modified and tailored to specific requirements, enabling users to customize it to suit their needs and integrate it into their workflows.

Numerous free software projects exist across various domains. Some prominent examples include the GNU/Linux OS, the Apache web server, the LibreOffice productivity suite, the GIMP image editor, the Firefox web browser, and the VLC media player.

Free software embodies the principles of user freedom, collaboration, and open access to source code. It empowers users to study, modify, and distribute software, fostering innovation, reliability, and customization. The free software movement has had a significant impact on the software industry, promoting a culture of transparency, cooperation, and the democratization of technology.

1.6 Linux distributions

One of the key strengths of Linux lies in its distribution ecosystem. Linux distributions, or "distros," are complete packages of the Linux kernel combined with various software components, utilities, and graphical interfaces. These distributions are built upon the Linux kernel and come with different software packages, desktop environments, and configuration options. Examples of popular Linux distributions include Ubuntu, Fedora, Debian, and CentOS. These distros cater to different user needs, ranging from beginner-

friendly desktop environments to specialized distributions for specific purposes like security or multimedia production. Here are some of the popular Linux distributions:

- Ubuntu: Ubuntu is one of the most widely used Linux distributions, known for its user-friendly interface and emphasis on ease of use. It offers a variety of flavors such as Ubuntu GNOME, Kubuntu (KDE desktop), Xubuntu (Xfce desktop), and Lubuntu (LXQt desktop).
- Fedora: Developed by the Fedora Project, sponsored by Red Hat, Fedora is known for its focus on the latest software and technologies. It serves as a cutting-edge distribution for both desktop and server environments.
- Debian: Debian is a highly respected and community-driven distribution known for its stability and adherence to the principles of free software. It serves as the foundation for several other distributions, including Ubuntu, Knoppix.
- CentOS: CentOS (Community Enterprise Operating System) is a distribution based on the source code of Red Hat Enterprise Linux (RHEL). It aims to provide a free and open-source alternative with long-term support for server deployments.
- Arch Linux: Arch Linux is a minimalist and lightweight distribution designed for experienced users who prefer to build their system from the ground up. It follows a rolling-release model, offering the latest updates and software packages.
- openSUSE: openSUSE is a community-driven distribution known for its stability and flexibility. It offers both the Leap edition, which focuses on stability, and the Tumbleweed edition, which provides rolling releases with the latest updates.
- Linux Mint: Linux Mint is a user-friendly distribution that aims to provide a polished and intuitive desktop experience. It is based on Ubuntu and offers different desktop environments like Cinnamon, MATE, and Xfce.
- Manjaro: Manjaro is an Arch Linux-based distribution that focuses on accessibility and user-friendliness. It offers a variety of desktop environments and aims to provide a hassle-free experience for both beginners and experienced users.
- Elementary OS: Elementary OS is a visually appealing distribution designed to provide a clean and user-friendly interface similar to macOS. It offers a customized desktop environment called Pantheon.
- Kali Linux: Kali Linux is a specialized distribution focused on penetration testing and ethical hacking. It comes preloaded with a wide range of tools for security testing and forensics.

These are just a few examples of the many Linux distributions available. Each distribution caters to different user needs, preferences, and use cases, ranging from general-purpose desktop systems to specialized server deployments or specific niches within the computing world. However, these distributions can be classified in four classes, which are:

- Debian: such as Ubuntu and Knoppix,
- Red Hat: such as CentOS, Scientific Linux,
- SLS/Slackware: such as SuSU Linux,
- Miscellany: in this class we find the rest of distribution.

1.7 Conclusion

The success of Linux has transcended its technical merits. It has become a symbol of the open-source movement, representing the values of collaboration, community, and shared knowledge. Linux has inspired countless individuals and organizations to embrace open-source principles, leading to the development of a vast array of free and open-source software across various domains. Linux and Innovation: Linux's impact on technological innovation is far-reaching. Its open nature encourages experimentation, enabling developers to create and modify software without constraints. The Linux kernel has served as the foundation for numerous groundbreaking projects, including Android, which revolutionized the smartphone industry, and the Docker containerization platform, which revolutionized application deployment.

1.8 Exercises

- On what type(s) of devices can Linux run?
- Based on today, how these OS platforms can be ranked in order of most to least commonly used in computing market: Linux, Windows, IBM System i, Mac OS.
- Give a definition of open source software.
- Why we should learn Linux.
- OS has a kernel, which can be seen as the most important part, what is an OS kernel? At what point is the kernel loaded into memory? How long does it stay in memory?
- Give an order to these layers' computer from top to down (user to hardware): OS kernel, device drivers, application software, OS services, shells, ROM Bios, OS utilities.
- In terms of speed from fast to slow, rank these devices: mainframe, server, smart phone, supercomputer, desktop computer.

2.1 Introduction

In Linux, you can interact with the system through two primary interfaces: the shell interface and the graphical user interface (GUI). The shell interface, also known as the command line interface (CLI) or terminal, allows you to interact with the operating system using text-based commands. There are various shell programs available in Linux, with Bash (Bourne Again Shell) being the most commonly used. However, Graphical User Interface (GUI) provides a visual way to interact with the operating system, using windows, icons, menus, and pointing devices like a mouse. It allows you to perform tasks by selecting options and clicking on objects. Figure 2.1 gives an example of CLI and GUI.

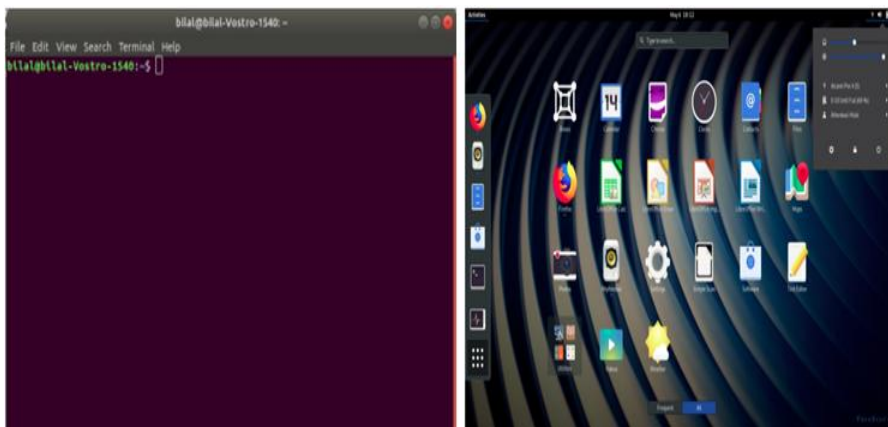


Figure 2.1: CLI and GUI interface.

The shell has some advantages, among them we find:

- **Powerful and flexible:** Shell commands provide extensive control over the system and allow you to perform complex tasks efficiently.
- **Automation:** Shell scripting allows you to automate repetitive tasks, creating scripts that execute a series of commands.
- **Remote access:** The shell interface can be accessed remotely over a network, making it useful for server administration.

On the other side GUI has also some advantages, for instance:

- **Ease of use:** GUIs are generally more user-friendly, especially for users who are not familiar with command-line interfaces.
- **Visual representation:** Graphical interfaces make it easier to visualize and understand information.
- **WYSIWYG (What You See Is What You Get):** GUI-based applications often provide a real-time representation of the output or result.

In this chapter, Bash shell will be examined. Some basic Linux commands will be introduced in order to see how to use some of Linux features. This chapter will cover

methods of obtaining help, Bash-specific features, comparison to other shells, and details on how the Bash interpreter works.

2.2 Shells' types

In Linux, there are several types of shells available, each with its own features, syntax, and capabilities. Here are some of the commonly used shells:

- **Bash (Bourne Again Shell):** Bash is the default shell for most Linux distributions. It is a widely used shell that provides a rich set of features, including command-line editing, history management, shell scripting capabilities, and support for variables and control structures. Bash is backward-compatible with the original Bourne shell (sh) and includes additional features from other shells like csh and ksh.
- **Zsh (Z Shell):** Zsh is an extended shell that offers a high level of customization and interactive features. It provides advanced tab completion, spelling correction, path expansion, and powerful scripting capabilities. Zsh is known for its extensive configuration options and is often used by power users who prefer a highly customizable shell.
- **Fish (Friendly Interactive Shell):** Fish is designed to be a user-friendly and interactive shell. It focuses on simplicity and ease of use, providing features like auto-completion, syntax highlighting, and suggestions as you type. Fish has a simplified syntax compared to other shells, making it more approachable for beginners.
- **Dash (Debian Almquist Shell):** Dash is a lightweight and fast shell that aims to be POSIX-compliant. It is commonly used as the default system shell on Debian-based distributions. Dash focuses on efficiency and is often preferred for scripting and system maintenance tasks.
- **Ksh (Korn Shell):** Ksh is an advanced shell that offers a combination of features from the Bourne shell (sh) and the C shell (csh). It provides powerful scripting capabilities, including advanced flow control structures, command history, and job control. Ksh is often used in commercial Unix environments.
- **Csh (C Shell):** Csh is a shell known for its C-like syntax and interactive features. It provides a set of powerful command-line editing capabilities, history management, and scripting features. Csh is less commonly used today but can still be found in some legacy systems.

These are many shells available in Linux. Each shell has its own set of features and advantages, so the choice of shell depends on personal preference, system requirements, and specific use cases. In this course we will focus on Bash shell, which is very popular and existed in different distribution such as Ubuntu.

2.3 Entering Linux commands

After logging into Linux and opening the shell (see figure 2.1 on left), you find a prompt to enter your command. It looks like this:

```
[username@hostname:current_directory]$
```

In Linux, the prompt in the shell interface represents the location where you can enter commands. The prompt typically consists of some text followed by a cursor indicating the current position for entering commands. The specific format of the prompt can vary depending on the shell being used and the customization settings. In general, the prompt in the Bash shell (the most commonly used shell in Linux) typically shows the following information:

- `username`: This is your username on the system.
- `hostname`: This is the name of the computer or system you are logged into.
- `current_directory`: This shows the current working directory, which represents your current location in the file system.
- `$`: The `$` symbol indicates that you are logged in as a regular user. If you see a `#` symbol instead, it means you are logged in as the root user (superuser) with administrative privileges.

The prompt can be customized to suit your preferences or to display additional information. You can modify the prompt by changing the value of the environment variable. The customization options are extensive, allowing you to include various variables, colors, and even add custom scripts or functions to the prompt.

It is worth to mention that the customization of the prompt is specific to the shell you are using. Different shells may have their own syntax and variables for customizing the prompt.

2.4 Command `man`

The `man` command in Linux is used to display the manual pages (documentation) for various commands, utilities, and system functions. To access the manual page for the Bash shell, you can use the following command:

```
man bash
```

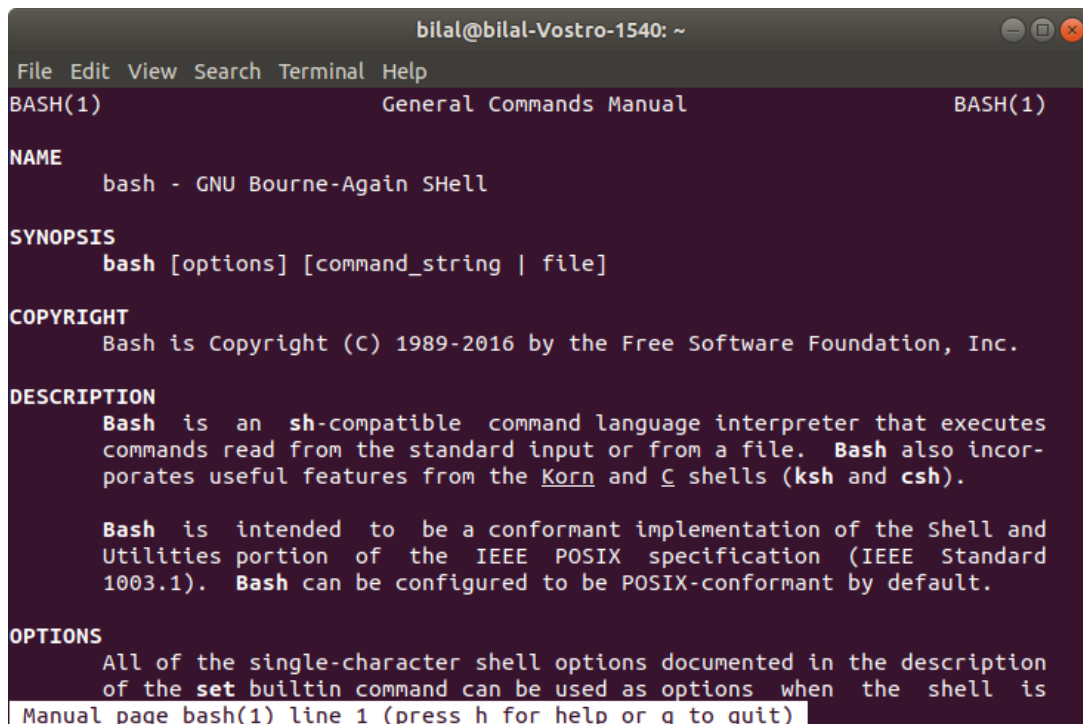
Running this command will open the manual page for Bash in your default pager program, which allows you to navigate and read the documentation. The Bash manual page provides comprehensive information about the shell, including its features, syntax, built-in commands, environment variables, shell options, and much more. It serves as a valuable reference for understanding and utilizing the capabilities of Bash.

Once you are in the manual page, you can navigate using standard pager commands. For example:

- Press the **Up Arrow** or **Down Arrow** keys to scroll through the manual page.
- Press the **Page Up** or **Page Down** keys to move forward or backward by a full page.
- Press **Q** to exit the manual page and return to the shell prompt.

You can also search for specific keywords within the manual page by typing “/” followed by the search term and then pressing **Enter**. To find the next occurrence of the search term, press **n**.

The Bash manual page is quite extensive, so it may take some time to explore and familiarize yourself with all the information it provides. However, the `man` command is really a helpful and you can save time to understand how to use Bash commands. It is a mandatory to learn how to read the Bash manual. Figure 2.2 show the results of entering the command: `man bash`



```
bilal@bilal-Vostro-1540: ~
File Edit View Search Terminal Help
BASH(1)                               General Commands Manual          BASH(1)
NAME
  bash - GNU Bourne-Again SHell
SYNOPSIS
  bash [options] [command_string | file]
COPYRIGHT
  Bash is Copyright (C) 1989-2016 by the Free Software Foundation, Inc.
DESCRIPTION
  Bash is an sh-compatible command language interpreter that executes
  commands read from the standard input or from a file. Bash also incor-
  porates useful features from the Korn and C shells (ksh and csh).

  Bash is intended to be a conformant implementation of the Shell and
  Utilities portion of the IEEE POSIX specification (IEEE Standard
  1003.1). Bash can be configured to be POSIX-conformant by default.
OPTIONS
  All of the single-character shell options documented in the description
  of the set builtin command can be used as options when the shell is
Manual page bash(1) line 1 (press h for help or q to quit)
```

Figure 2.2: Results of `man Bash` command.

2.5 Linux commands

The Bash shell provides a wide range of built-in commands and utilities that you can use to interact with the Linux system and perform various tasks. There are several simple commands that can give you information about the user of the system, current location or directory, architecture, etc. Among these commands, we have:

- Commands about the user of the system:
 - `whoami`—output the user’s username
 - `pwd`—output the current working directory
 - `hostname`—output the name of the host (this will either be `localhost.localdomain` or the IP address or IP alias of the machine logged in to)
- File and Directory Operations:
 - `ls`—list the contents of the current working directory
 - `cd`—Change the current directory

- `pwd`—Print the current working directory
- `mkdir`—Create a new directory
- `rm`—Remove files and directories
- `cp`—Copy files and directories
- `mv`—Move or rename files and directories
- Text Processing:
 - `cat`—Concatenate and display the contents of files
 - `grep`—Search for patterns in files
 - `head`—Display the first lines of a file
 - `tail`—Display the last lines of a file
 - `sort`—Sort lines of text
 - `cut`—Extract specific fields from lines
 - `sed`—Stream editor for manipulating text
 - `awk`—Powerful text processing tool for extracting and manipulating data.
- File Viewing and Editing:
 - `less`— View files with pagination and search capabilities
 - `nano`— A simple command-line text editor
 - `vi` or `vim`—Advanced text editors with powerful editing features
 - `tee`—Read from standard input and write to files and the standard output simultaneously
- Process Management:
 - `ps`—View running processes
 - `top`—Monitor system processes in real-time
 - `kill`—Terminate or send signals to processes
 - `bg`—Run a process in the background
 - `fg`—Bring a background process to the foreground
 - `nohup`—Run a command immune to hangups and keep it running even after logout
- System Information and Control:
 - `uname`—Print system information
 - `df`—Display disk space usage
 - `du`—Estimate file and directory sizes
 - `free`—Display memory usage
 - `uptime`—Show system uptime
 - `shutdown`—Shutdown or restart the system
 - `arch`—output basic information about your computer’s hardware (architecture)
 - `passwd`—Allow the user to change passwords, or if specified with a username, allow root to change the specified user’s password.

It is possible to enter multiple commands from the prompt by separating them by a semicolon. Here is an example of the result of some commands:

```
bilal@bilal-Vostro-1540:~$ uname
Linux
```

```
bilal@bilal-Vostro-1540:~$ arch
x86_64
bilal@bilal-Vostro-1540:~$ who
bilal      :0                2023-06-03 10:07 (:0)
```

Linux commands typically require arguments, which can be classified into two main types: options and parameters. Options are user-defined modifiers that alter the behavior of a command, enabling it to perform tasks differently. For instance, the `ls` command offers various options to display additional information beyond a basic list of items in the current directory. For example, the option `"-l"` (hyphen lowercase L) generates a "long" listing that includes detailed information about each entity, going beyond just their names.

Parameters in Linux commands typically consist of file names, directory names, or a combination of both. These parameters serve to specify the target entity on which the command should operate. In the context of a Linux command, the format generally follows a predefined structure:

```
command [option(s)] [parameter(s)]
```

where `[]` means it is an option.

As an example, to view the long listing of a specific file named `file.txt`, you can execute the following command:

```
ls -l file1.txt
```

A significant number of Linux commands allow for multiple parameters. The previous instruction could have applied to multiple files with equal ease. This scenario would occur if the command were to be executed on several files simultaneously. For instance:

```
ls -l file1.txt file2.txt file3.txt file4.txt
```

These are just a few examples of the numerous commands available in the Bash shell. You can explore and discover more commands by referring to the Bash manual (`man bash`) or using the `help` command to see built-in Bash shell commands. Additionally, many commands have their own manual pages accessible through the `man` command (e.g., `man ls` for the `ls` command).

2.6 Bash features

The Bash shell, being a powerful and widely used shell in Linux, comes with several notable features that enhance its functionality and usability. Here are some key features of the Bash shell:

- **Command History:** Bash keeps a history of previously executed commands, allowing you to recall and reuse them easily. You can navigate through the `command history` using the arrow keys.
- **Tab Completion:** Bash provides tab completion, which automatically completes commands, file names, and directories by pressing the Tab key. It saves time and helps prevent typing errors.

- **Shell Scripting:** Bash is a fully-featured scripting language, enabling you to write shell scripts to automate tasks, perform complex operations, and create custom command sequences.
- **Environment Variables:** Bash allows you to define and use environment variables to store information such as paths, configurations, and custom settings. These variables can be accessed and modified within shell scripts and command-line operations.
- **Job Control:** Bash provides job control capabilities, allowing you to manage multiple running processes. You can start processes in the background, switch between foreground and background tasks, and suspend or resume processes using `bg` or `fg` commands.
- **Command Substitution:** Bash supports command substitution, which allows the output of one command to be used as input for another. It is denoted by using backticks (```) or the `$ (. . .)` syntax.
- **Redirection and Pipes:** Bash provides various operators for redirecting input and output streams. You can redirect command output to files (`>`), append output to files (`>>`), and redirect input from files (`<`). Additionally, you can use pipes (`|`) to connect the output of one command as input to another command.
- **Customization:** Bash can be customized to suit individual preferences. You can modify the shell prompt, set aliases for frequently used commands, define custom functions, and customize various settings using the `.bashrc` file in your home directory.
- **Process Substitution:** Bash supports process substitution, which allows the output of a command or process to be treated as a file. It is denoted by using the `<(. . .)` or `>(. . .)` syntax.
- **Extended Pattern Matching:** Bash offers extended pattern matching capabilities through the `extglob` option. It allows you to use more advanced patterns for matching filenames and performing pattern-based manipulations.

We have presented some features of the many features available in the Bash shell. Bash's extensive capabilities and flexibility make it a popular choice for both interactive use and shell scripting in the Linux environment.

2.7 Bash interpreter

The Bash interpreter, also known as the "Bourne Again Shell," is the default shell and interpreter for most Linux distributions. It is an enhanced version of the original Bourne shell (`sh`) and provides a powerful and feature-rich environment for executing commands and running shell scripts. The Bash interpreter reads and executes commands written in the Bash scripting language, which is a superset of the original Bourne shell syntax. It supports a wide range of features, including variables, command substitution, flow control structures (if-else statements, loops), functions, and more.

When you run a Bash script, the Bash interpreter parses and executes each line of the script in sequential order. It performs variable substitutions, executes commands, and handles control structures according to the script's logic. The interpreter can also handle user input, process signals, and manage environment variables. In addition to interactive use, the Bash interpreter is widely used for writing shell scripts to automate tasks and create complex command sequences. Bash scripts can be executed directly from the command line or scheduled as cron jobs.

Finally, the Bash interpreter is a fundamental component of the Linux command-line environment, providing a versatile and powerful scripting and interactive shell experience.

2.8 Conclusion

Fundamental concepts about shell have been presented in this chapter. There are many shells for Linux and each one has its own features. However, the most popular one is Bash. It has been implemented in different distribution and is a powerful shell. With Bash the user can enter commands through the prompt according to the task. Some of these commands have been illustrated. Commands may have options and parameters. It has been mentioned that man command is very helpful and it is a mandatory to learn how to read the output of this command.

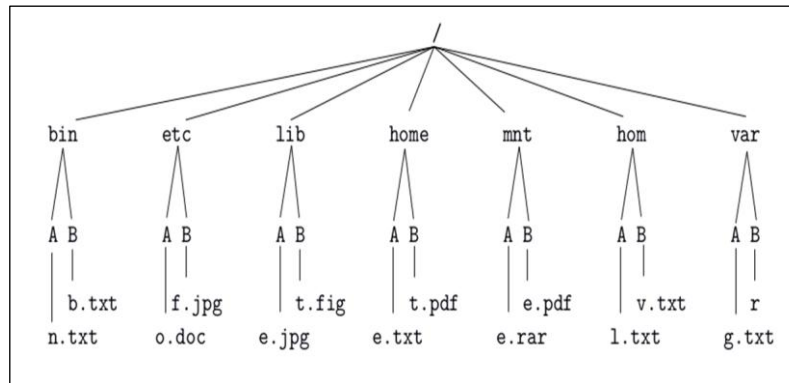
2.9 Exercises

- Why do we refer to the user's environment as a shell?
- Given the following prompt, answer the questions below.
[zappaf@frenchie /etc]#
 - Who is logged in?
 - Is the user logged in as themselves or root?
 - What is the current working directory?
 - What is the hostname?
- If "Mouloud" is logged in to a Linux computer on console and user "Smail" is logged into the computer remotely from IP address 1.2.3.4. What will be the output of the command `who`?
- We have a text file "phone.txt" below. Each line is of the form "last name first name phone number". Words are separated by tabs. Answer the following questions, each time using a shell command line:

```
AISSAOUI CHABANE 0381818585
AITMESSAOUD YOUSRA 0478858689
AKKOUCHE SIHAM 04961868957
AKKOUCHE HASSINA 0685987456
BENSAADIA SAID 0658769458
BERNOU KAMEL 0131986258
...
```

- Display the number of people in "phone.txt".

- Display all the lines concerning the "AKKOUCHE".
- Display all the lines not concerning the "AKKOUCHE".
- Display the phone number (without first and last name) of the first "AKKOUCHE" appearing in "phone.txt".
- Display the phone number (without first and last name) of the first "AKKOUCHE" in alphabetical order.
- What is the sequence of commands needed to delete the "e.txt" file.



3.1 Introduction

One of the key advantages of the Linux platform is its extensive development by the open-source community. This has resulted in not only the availability of the operating system itself at no cost but also a vast array of free application software for Linux. In this chapter, our focus is dedicated to exploring some of the widely adopted and valuable Linux applications.

In this chapter, our primary focus will be on two commonly used text editors: vi and emacs. The vi editor is included as part of the Linux installation, while the availability of the emacs editor may vary depending on your specific installation. We will also provide a brief overview OpenOffice, which is one of the most popular productivity software suites for Linux. Our exploration of OpenOffice will be limited, as it assumes that you will find it relatively straightforward to navigate based on your prior experience with other productivity software like Microsoft Office. Additionally, we will briefly examine LaTeX, a text-based word processor that enjoys significant popularity among Linux users. LaTeX stands out as a powerful word processor that differs from typical GUI-based WYSIWYG (what you see is what you get) word processors. It harkens back to the early days of word processing, where commands are embedded within the text itself.

3.2 Text editors

Linux offers a variety of text editors that cater to different needs and preferences. Here are some popular text editors in the Linux ecosystem:

- Vi/Vim: Vi (Visual Editor) is a classic and widely used text editor that comes preinstalled on most Linux distributions. Vim (Vi Improved) is an enhanced version of Vi with additional features and functionality. Both Vi and Vim are known for their powerful command-line interface and extensive customization options.
- Emacs: Emacs is a highly customizable and extensible text editor that provides a rich set of features and capabilities. It offers a wide range of functionalities, including text editing, file management, and integrated development environment (IDE) capabilities. Emacs uses its own scripting language (Emacs Lisp) for customization and extensibility.
- Nano: Nano is a user-friendly and straightforward text editor that is often recommended for beginners. It offers a simple and intuitive interface with basic editing functionalities. Nano is known for its ease of use and minimal learning curve.
- Gedit: Gedit is the default text editor for the GNOME desktop environment. It provides a clean and user-friendly interface with essential features for text editing, syntax highlighting, and plug-in support. Gedit is often preferred by users who prefer a graphical user interface (GUI) for their text editing tasks.

- Sublime Text: Sublime Text is a popular cross-platform text editor that is also available for Linux. It offers a sleek and customizable interface with powerful editing features, multiple selection capabilities, and extensive plug-in support. Sublime Text is known for its performance and responsiveness.
- Atom: Atom is another widely used text editor that is open-source and highly customizable. It provides a modern and feature-rich interface with built-in package management for adding functionalities. Atom is developed by GitHub and offers a vibrant ecosystem of themes, packages, and extensions.
- Visual Studio Code: Visual Studio Code (VS Code) is a powerful and versatile code editor that supports multiple programming languages. It offers a wide range of features, including IntelliSense code completion, debugging, version control integration, and an extensive library of extensions. VS Code is widely adopted by developers for various programming tasks.

These are just a few examples of text editors available in the Linux ecosystem. Each text editor has its own set of features, interface, and target audience. The choice of a text editor often depends on personal preferences, workflow requirements, and the specific tasks at hand.

3.2.1. Vi/Vim

The VI text editor, often referred to as Vim (Vi IMproved), is a powerful and widely used text editor in the Linux environment. It is a modal editor, meaning it operates in different modes for different functions. Here's an overview of the VI text editor:

- Modes:
 - Normal Mode: The default mode for navigation, copying, pasting, and performing operations on text.
 - Insert Mode: Allows you to insert and edit text.
 - Command Mode: Used for executing commands such as saving files, searching, and replacing.
- Keybindings:
 - Navigation: Use the arrow keys or **h**, **j**, **k**, **l** for left, down, up, and right respectively.
 - Editing: Press **i** to enter insert mode and start editing text.
 - Saving and Quitting: In command mode, use **:w** to save changes and **:q** to quit. To save and quit together, use **:wq**.
 - Searching: In command mode, type **/** followed by the search term and press Enter to search forward. Use **?** for reverse search.
- Features:
 - Syntax Highlighting: Vim provides syntax highlighting for various programming languages, making code more readable.
 - Split Windows: You can split the editor window into multiple panes for working on different parts of a file simultaneously.

- Plugins and Customization: Vim supports a vast array of plugins and offers extensive customization options through a configuration file called `.vimrc`.
- Command-Line Options:
 - Opening a File: Use `vim <filename>` to open an existing file. If the file doesn't exist, Vim creates a new file with that name.
 - Opening in Read-Only Mode: Use `vim -R <filename>` to open a file in read-only mode.
 - Editing Multiple Files: Use `vim <file1> <file2>` to open multiple files in separate buffers within Vim.

Vim has a steep learning curve but offers incredible power and efficiency once you become familiar with its features and commands. It is highly customizable and has a strong and dedicated user community, making it one of the most popular text editors among Linux users and developers. Figure 3.1 shows VI interface from Ubuntu system.

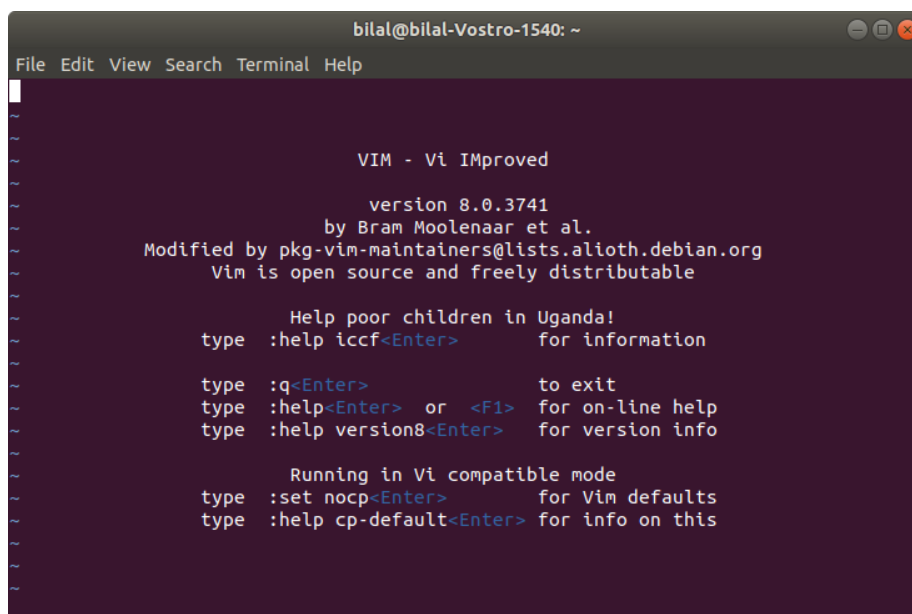


Figure 3.1: VI interface.

3.2.2. Emacs

The Emacs text editor is a powerful and extensible text editing environment commonly used in the Linux ecosystem. It provides a wide range of features and customization options. Here's an overview of Emacs:

- Modes and Buffers:
 - Text Editing: Emacs provides a comprehensive set of keybindings and commands for efficient text editing.
 - Major Modes: Emacs supports major modes tailored for different programming languages, offering syntax highlighting, indentation, and specialized commands.

- Buffer Management: Multiple files or buffers can be opened simultaneously, allowing you to switch between them easily.
- Customization:
 - Emacs Lisp: Emacs can be extended and customized using Emacs Lisp, a powerful and flexible programming language. Users can write their own functions and modify existing ones to suit their workflow.
 - Initialization File: Emacs loads an initialization file called `.emacs` or `init.el`, where users can configure settings, define keybindings, and load additional packages.
- Features:
 - Split Windows: Emacs supports dividing the editor window into multiple panes for simultaneous editing and viewing.
 - Version Control Integration: Emacs integrates with various version control systems such as Git, allowing you to perform version control operations within the editor.
 - Interactive Shell: Emacs can run an interactive shell, providing a command-line interface within the editor itself.
 - Integrated Development Environment (IDE) Features: Emacs offers features like code navigation, code completion, and integrated debugging for various programming languages.
- Package Manager:
 - Package repositories like MELPA and ELPA provide a vast collection of community-contributed packages, extending Emacs with additional functionality, such as project management tools, additional programming language support, and more.
- Keybindings:
 - Emacs uses its own set of keybindings. Commonly used key combinations include Ctrl and Alt (or Meta) in combination with letters or function keys.

Emacs has a steep learning curve, but its versatility and extensibility make it a popular choice for power users and developers. It offers a highly customizable environment, allowing users to tailor it to their specific needs. Emacs has a dedicated user community and continues to be widely used for both text editing and programming tasks in the Linux ecosystem. The following figure illustrates Emacs workspace.

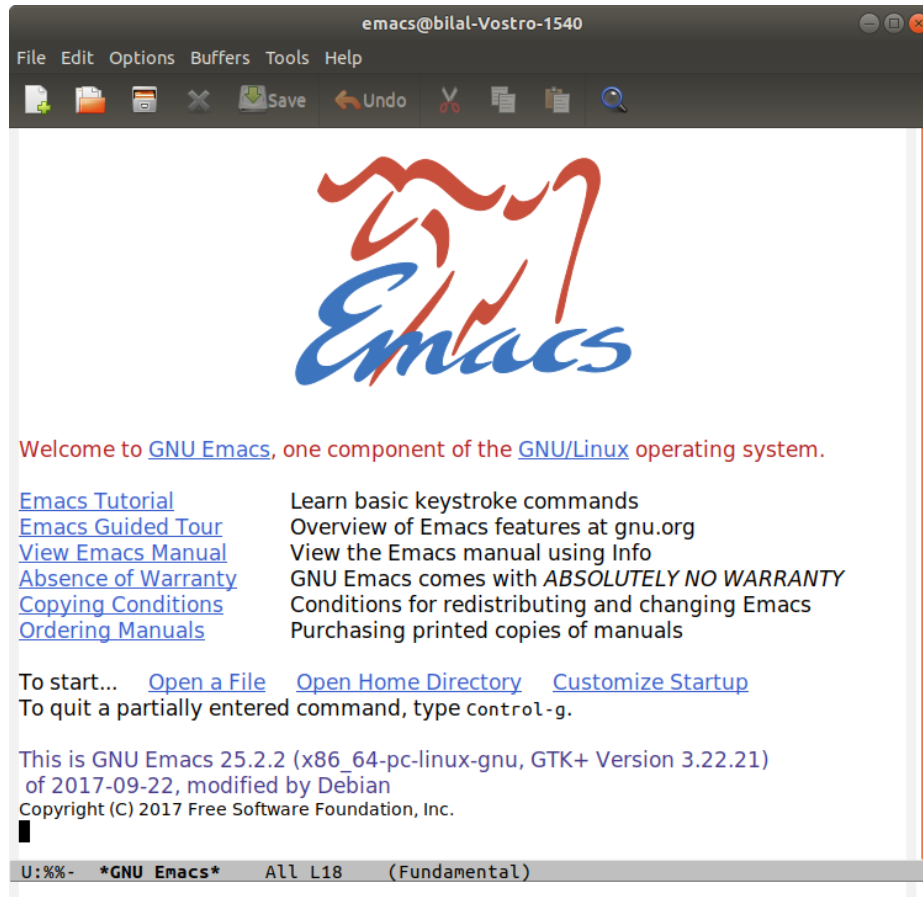


Figure 3.2: Emacs interface.

3.3 Productivity software

Linux offers a wide range of productivity software that can enhance your efficiency and help you accomplish various tasks. Here are some popular productivity software options available for Linux:

- Office Suites:
 - LibreOffice: A comprehensive office suite including word processing, spreadsheets, presentations, databases, and more. It is compatible with Microsoft Office file formats.
 - Apache OpenOffice: Another full-featured office suite offering similar applications as LibreOffice.
 - OnlyOffice: An open-source office suite with collaboration features and support for cloud integration.
- Note-taking and Document Management:
 - Joplin: A note-taking application with support for Markdown, file attachments, synchronization, and encryption.
 - Zotero: A powerful research tool for managing citations, references, and bibliographies.
 - Calibre: An ebook management tool for organizing and converting ebook formats.

- Task and Project Management:
 - Todoist: A popular task management application with cross-platform support and synchronization.
 - Taskwarrior: A command-line task management tool with flexible features and support for advanced task organization.
 - Redmine: A web-based project management and issue tracking system suitable for team collaboration.
- Communication and Collaboration:
 - Thunderbird: An email client with advanced features, including customizable filters, encryption, and add-on support.
 - Slack: A team communication and collaboration platform for real-time messaging, file sharing, and integration with other tools.
 - Mattermost: A self-hosted team communication platform that provides secure messaging and collaboration features.
- Time Tracking and Productivity Tools:
 - RescueTime: A time tracking and productivity tool that monitors your computer usage and provides insights into your productivity habits.
 - Pomodoro Timer: Various Pomodoro technique timers are available on Linux to help you manage your time and stay focused.
- Graphics and Design:
 - GIMP: A powerful image editing and manipulation software, comparable to Adobe Photoshop.
 - Inkscape: A vector graphics editor suitable for creating and editing illustrations, logos, and diagrams.

These are just a few examples of productivity software available for Linux. The Linux ecosystem offers a vast selection of applications across different categories, enabling users to find tools that cater to their specific productivity needs.

3.4 Latex

LaTeX is a typesetting system widely used for creating high-quality documents, especially in the academic and scientific communities. It provides excellent control over document formatting, mathematical equations, citations, and cross-referencing. LaTeX is available for Linux systems and can be installed and used with ease. Here's an overview of using LaTeX in Linux:

- Installation:
 - LaTeX distribution: Install a LaTeX distribution package such as *TeX Live* or *MiKTeX*. These distributions include all the necessary components, including the LaTeX compiler (usually `pdflatex` or `latex`) and various packages.
 - Package Manager: Many Linux distributions provide package managers to install LaTeX and related packages. For example, on Ubuntu, you can use the `apt` package manager to install the `texlive` package.
- Text Editor:

- Choose a text editor: LaTeX documents are plain text files, so you can use any text editor to write LaTeX code. Popular choices include Emacs, Vim, Atom, Sublime Text, and VS Code.
- LaTeX Editors: Alternatively, you can use LaTeX-specific editors such as TeXstudio, Kile, or Overleaf, which provide dedicated features and functionalities for writing and compiling LaTeX documents.
- Workflow:
 - Write LaTeX code: Create a new .tex file and write your document using LaTeX markup. LaTeX uses commands and environments to structure the document, format text, and include elements such as equations, tables, and images.
 - Compile the document: Use the LaTeX compiler to process your .tex file and generate a PDF output. The compilation process may involve running the compiler multiple times to resolve references and update the document's layout.
 - View and refine: Open the generated PDF to view the final document. Make any necessary changes to the LaTeX code, recompile, and iterate until you are satisfied with the output.
- Resources:
 - Documentation: LaTeX has extensive documentation available, including beginner's guides, reference manuals, and online resources. The official LaTeX project website (<https://www.latex-project.org/>) provides a wealth of information.
 - Templates: Numerous LaTeX templates are available for various types of documents, such as articles, reports, theses, and presentations. You can find templates online or use built-in templates provided by LaTeX editors.

LaTeX provides a powerful and flexible system for creating professional documents, especially those containing complex mathematical equations, scientific notations, and bibliographies. Its wide adoption and robustness make it a preferred choice for many Linux users in academia and research.

In LaTeX, a document is structured using various elements and commands to define the overall layout, formatting, and content organization. Here's a typical structure of a LaTeX document:

- Document Class:
 - Begin with the document class declaration, which defines the type of document you are creating. Common document classes include article, report, book, and beamer (for presentations).
 - Example: `\documentclass{article}`
- Preamble:
 - The preamble section comes after the document class declaration and before the `\begin{document}` command.
 - In the preamble, you can load packages, set document-wide settings, define custom commands, and configure the document's appearance.
 - Example: Setting the font size, loading packages, defining custom commands.

- **Begin Document:**
 - The `\begin{document}` command marks the beginning of the document's content.
 - After this command, you start writing the actual text and inserting various elements like sections, paragraphs, equations, figures, tables, etc.
- **Title and Author:**
 - Specify the title and author of the document using the `\title{}` and `\author{}` commands.
 - Example: `\title{My Document} \author{Bilal SAOUD}`
- **Date:**
 - Optionally, include the date using the `\date{}` command.
 - Example: `\date{\today}` (to include the current date)
- **Abstract (for certain document classes):**
 - In some document classes, such as article or report, you can include an abstract using the `abstract` environment.
 - Example: `\begin{abstract} This is the abstract of the document. \end{abstract}`
- **Body:**
 - The body of the document contains the main content.
 - You can structure the document using various commands like `\section{}`, `\subsection{}`, `\paragraph{}`, etc., to create sections and subsections.
 - Include text, mathematical equations, lists, tables, figures, citations, and other elements as needed.
- **End Document:**
 - The `\end{document}` command marks the end of the document.

Figure 3.4 illustrates an example of a document in LaTeX.

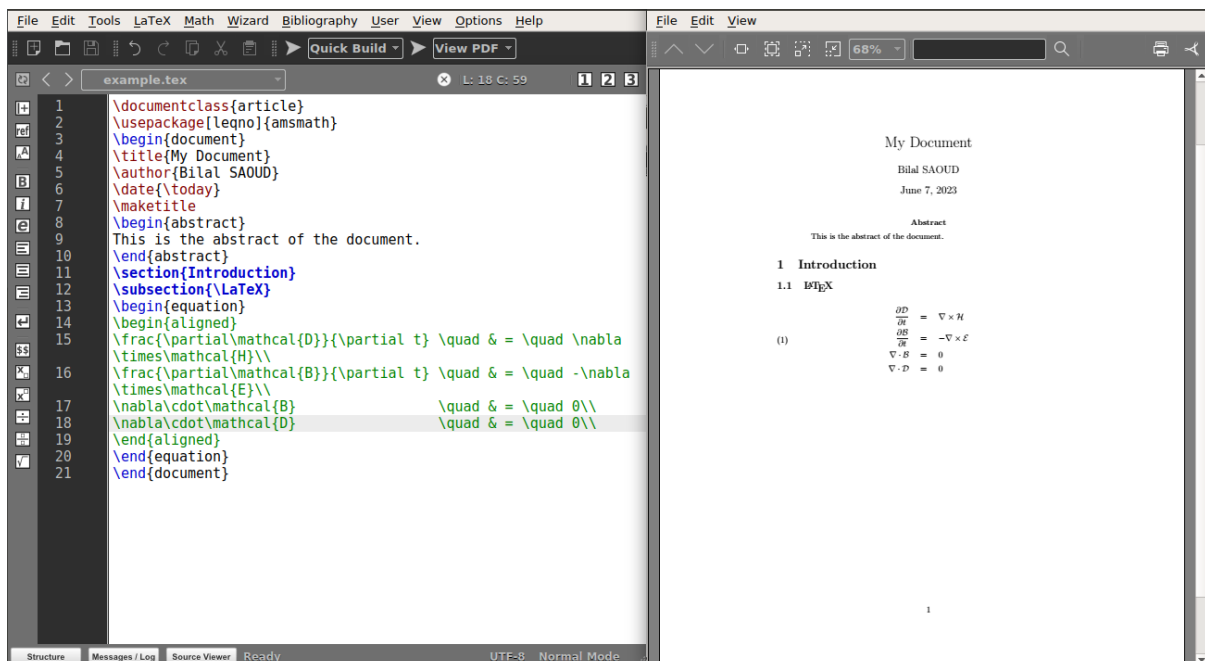


Figure 3.4: Example of LaTeX document.

The structure outlined above provides a basic framework for a LaTeX document. However, depending on the document class and your specific requirements, you may need to use additional commands and environments to format and organize your content effectively. It is also common to use additional packages to extend LaTeX's capabilities for specific tasks or document types.

3.5 Encryption software

In this section, we will explore various encryption software options in Linux, with a focus on OpenSSL and other related tools. However, before delving into the software details, let's take a moment to understand what encryption is, why it is important, and how we can effectively utilize it.

3.4.1. What is encryption?

Encryption is a technique used to transform data, represented as a sequence of characters (such as ASCII, Unicode, binary, etc.), into a modified form using a specific code or algorithm. This encoded message is designed to be difficult to comprehend if intercepted by unauthorized parties. Encryption involves two main processes: encryption and decryption. Encryption is the process of converting information into a coded form, while decryption is the reverse process of restoring the original form a coded message.

3.4.2. Openssl

OpenSSL is a widely used open-source software library that provides support for secure communications and cryptographic functions. It is available on Linux and various other platforms. Here are some key aspects of OpenSSL in Linux:

- **Cryptographic Functions:**
 - OpenSSL offers a wide range of cryptographic functions, including encryption, decryption, hashing, digital signatures, and key management.
 - It supports various encryption algorithms such as AES (Advanced Encryption Standard), DES (Data Encryption Standard), RSA (Rivest-Shamir-Adleman), and more.
 - OpenSSL also provides support for symmetric and asymmetric encryption, allowing you to choose the appropriate encryption method based on your requirements.
- **SSL/TLS Protocols:**
 - OpenSSL implements the SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols, essential for secure communication over networks.
 - It allows applications to establish encrypted connections, ensuring data confidentiality and integrity.
 - OpenSSL supports different versions of SSL/TLS, including SSLv3, TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3.
- **Command-Line Tools:**
 - OpenSSL provides a set of command-line tools that allow you to perform various cryptographic operations.

- The openssl command-line tool is versatile and can be used for tasks such as generating key pairs, creating certificates, encrypting and decrypting files, calculating message digests, and testing SSL/TLS connections.
- Library Integration:
 - OpenSSL is widely used as a library in various software applications and frameworks.
 - It provides a robust and reliable foundation for incorporating secure communication and encryption capabilities into Linux applications.
 - Developers can utilize the OpenSSL library to implement secure network protocols, secure file transfers, cryptographic operations, and more.
- Security Enhancements:
 - OpenSSL undergoes continuous development and improvement to address security vulnerabilities and enhance the overall security of cryptographic functions.
 - Regular updates and patches are released to ensure the library's integrity and mitigate potential risks.

OpenSSL is a powerful and flexible cryptographic toolkit that plays a crucial role in securing communications and data in Linux systems. It is utilized by a wide range of applications, including web servers, email clients, VPNs, and more. The availability of command-line tools and its integration as a library make OpenSSL a popular choice for developers and system administrators seeking to incorporate encryption and security features into their Linux-based solutions.

3.6 Network software

Linux offers a wide range of network software that facilitates various networking tasks and functionalities. Here are some popular network software options available for Linux:

- Network Configuration and Management:
 - NetworkManager: A network management tool that provides a graphical interface for configuring and managing network connections.
 - systemd-networkd: A lightweight network configuration daemon that allows you to manage network settings using simple configuration files.
 - ifconfig: A command-line tool for managing network interfaces, assigning IP addresses, enabling or disabling interfaces, and more.
- Network Monitoring and Troubleshooting:
 - Wireshark: A powerful network protocol analyzer that captures and analyzes network traffic, allowing you to troubleshoot and diagnose network issues.
 - tcpdump: A command-line packet analyzer that captures and displays network traffic in real-time or from pcap files.
 - nmap: A network scanning tool used to discover hosts, services, and open ports on a network.
- Firewall and Security:
 - iptables: A versatile firewall utility that allows you to set up rules to filter and manipulate network traffic.

- UFW (Uncomplicated Firewall): A user-friendly command-line interface to iptables, providing an easier way to manage firewall rules.
- OpenVPN: An open-source VPN (Virtual Private Network) solution that provides secure communication over the internet.
- File Transfer and Remote Access:
 - OpenSSH: A suite of secure network utilities that enables secure remote login, file transfer (scp and sftp), and secure remote execution (ssh).
 - FileZilla: A popular FTP (File Transfer Protocol) client that allows you to transfer files to and from remote servers.
 - Rsync: A powerful file synchronization and transfer utility that efficiently transfers and synchronizes files between local and remote systems.
- Network Services and Protocols:
 - Apache HTTP Server: A widely used web server software for hosting websites and serving web content.
 - DNS (Domain Name System) Servers: Linux provides various DNS server software options like BIND (Berkeley Internet Name Domain), PowerDNS, and Unbound.
 - DHCP (Dynamic Host Configuration Protocol) Servers: Software like ISC DHCP server allows you to manage IP address allocation and configuration for network devices.
- Network Monitoring and Management:
 - Nagios: A popular network monitoring system that monitors network services, hosts, and performance.
 - Cacti: A network graphing solution that provides visual representations of network data and performance metrics.
 - Zabbix: An enterprise-level monitoring solution that offers real-time monitoring, alerting, and visualization of network resources.

These are just a few examples of the network software available for Linux. The Linux ecosystem provides a wide range of tools and solutions to meet various networking needs, including configuration, monitoring, security, and remote access.

3.7 Conclusion

In conclusion, the Linux platform offers numerous advantages due to its extensive development by the open-source community. Not only is the operating system itself freely available, but there is also a vast selection of free application software for Linux. Throughout this chapter, we have explored some widely adopted and valuable Linux applications. Our focus primarily revolved around two commonly used text editors, VI and Emacs. While VI is included as part of the Linux installation, the availability of Emacs may vary. We also provided an overview of OpenOffice, one of the most popular productivity software. Furthermore, we briefly explored LaTeX, a text-based word processor highly favored by Linux users.

Linux ecosystem offers a wide range of powerful applications that cater to diverse user needs, thanks to the collaborative efforts of the open-source community. Whether it's text

editors, productivity software, or specialized tools like LaTeX, Linux provides a rich software landscape for users to explore and leverage in their computing endeavors.

3.8 Exercises

- In vi, what is the difference between the command 'r' and 'i'?
- In vi, what is the difference between the command 'o' and 'O'?
- In vi, what is the difference between the command 'w' and 'W'?
- In vi, what is the difference between the command 'e' and 'E'?
- In vi, you are editing a document and currently on line 21. How would you reach line the last line of the document?
- Why might you want to use LaTeX rather than a WYSIWYG word processor?
- Given the following LaTeX definition for a table, give its result?

```
\begin{tabular}{|l|l|l|l|}  
    Item 1 & Item 2 \\  
    Item 3 & Item 4 \\  
    Item 5 \\  
    & Item 6 \\  
\end{tabular}
```

- What is TeX? How does it differ from LaTeX?
- Using openssl, what are the steps that you would need to generate a private key?
- Using openssl, what are the steps that you would need to generate a public key?

5.1 Introduction

In this chapter, we will delve into the fundamentals of shell scripting, exploring the various features, syntax, and techniques that make it an essential skill for Linux users and administrators. Whether you are a beginner looking to grasp the basics or an experienced user seeking to enhance your scripting abilities, this chapter will provide valuable insights and practical examples to help you master the art of shell scripting.

5.2 Simple scripting

Shell scripting is a powerful tool that allows you to automate tasks, customize your Linux environment, and unleash the true potential of the command-line interface. Shell scripts are essentially text files containing a series of commands that are executed sequentially when the script is run.

A shell script starts by Shebang. The shebang line, also known as the hashbang, is the first line of a shell script that specifies the interpreter to use. For Bash scripts, the shebang line typically begins with:

```
#!/bin/bash
```

Shell scripts can execute commands and capture their output by enclosing them within backticks "" or using the "\$()" syntax. This allows for dynamic data processing within the script. For instance, a script to see the content of a directory will be:

```
#!/bin/bash
ls
```

this script is saved in file with a name "file.sh". In order to run this script we should write on the shell prompt the following commands:

```
$ chmod +x file.sh
$ ./file.sh
```

The first command will be used just once. The command "chmod +x" is used in Linux and Unix-based systems to change the permissions of a file or script to make it executable. The "chmod" command stands for "change mode" and allows you to modify the permissions of a file or directory. The "+x" option is used to add the execute permission to the file. By adding the execute permission, you allow the file to be executed as a program or script. Finally, to run the script we use the command "./".

5.3 Variables, assignments and parameters

Shell variables are used to store and manipulate data within a shell script. They allow you to assign values to named variables and use those values throughout the script. Shell variables can hold various types of data, including strings, numbers, and file names. Here are some important aspects of shell variables:

- Variable assignment: Variables are assigned values using the "=" operator. For example:
name="Mohamed"
age=23

In the above example, the variable "name" is assigned the value "Mohamed", and the variable "age" is assigned the value 23.

- Variable expansion: To access the value of a variable, you use the "\$" symbol followed by the variable name. For example:

```
echo "My name is $name"
```

The above statement will output "My name is Mohamed" by expanding the value of the "name" variable.

- Environment variables: Shell scripts can access environment variables, which are predefined variables that contain information about the shell environment. Environment variables are denoted by uppercase names and can be accessed like regular variables. For example, the "PATH" environment variable contains the list of directories where executable files are located.
- Command substitution: Shell scripts can assign the output of a command to a variable using command substitution. Command substitution is performed by enclosing the command within backticks "`" or using the "\$()" syntax. For example:

```
current_date=$(date +%Y-%m-%d)
```

The above command assigns the current date in the "YYYY-MM-DD" format to the "current_date" variable.

- Readonly variables: You can make a variable read-only by using the "readonly" command. Once a variable is marked as readonly, its value cannot be changed. For example:

```
readonly version="1.0"
```

The above statement makes the "version" variable read-only.

- Local variables: By default, variables in a shell script have global scope, meaning they can be accessed from anywhere in the script. However, you can declare variables as local using the "local" keyword within functions. Local variables are only accessible within the function where they are defined.

Shell variables provide a flexible way to store and manipulate data in shell scripts. They are instrumental in performing calculations, storing user input, managing file names, and controlling the flow of the script based on conditions.

5.4 Input and output

In shell scripting, input and output refer to the communication between the shell script and the user, as well as with external files and programs. Shell scripts provide various mechanisms for handling input and output, allowing for interaction, data retrieval, and data manipulation. Here are some key aspects of input and output in shell scripting:

- Standard Input (stdin): Standard input is the default source of input for a shell script. It represents the data that is provided to the script, typically from the keyboard or another program. The script can read input from stdin using commands like read or by reading input from files or pipes.

- **Command-line arguments:** Command-line arguments are input values that are passed to a shell script when it is executed. They allow users to provide specific values or parameters to customize the behavior of the script. Command-line arguments can be accessed within the script using variables like `$1`, `$2`, and so on, where `$1` represents the first argument, `$2` represents the second argument, and so on.
- **Standard Output (stdout):** Standard output is the default destination for the script's output. It represents the information or results that the script generates. The script can output data to stdout using the `echo` command or by redirecting the output of commands or scripts to stdout.
- **Standard Error (stderr):** Standard error is a separate output stream that is used to capture error messages or warnings generated by the script. It allows the differentiation of normal output and error messages. By default, stderr is displayed on the terminal along with stdout. However, it can be redirected to a file or suppressed using redirection operators.
- **Redirection:** Shell scripting provides redirection operators to control where input comes from and where output goes. The `>` operator is used to redirect output to a file, overwriting its contents if it exists, while `>>` appends output to a file. The `<` operator is used to redirect input from a file. Redirection can also be used to redirect stdout to stderr or vice versa.
- **Pipes:** Pipes, represented by the `|` symbol, allow for the redirection of output from one command to another as input. This enables chaining of commands and allows the output of one command to be processed by another command without writing to intermediate files.
- **File Input/Output:** Shell scripts can read and write data to files using various commands like **cat**, **grep**, **sed**, and **awk**. These commands allow the script to read and process data from files, as well as write output to files.
- **Here documents:** Here documents are a way to provide multiline input to a command or script within the script itself. They allow embedding of large blocks of text directly in the script without the need for external files.

Input and output handling in shell scripting is crucial for creating interactive scripts, processing data, and generating useful output. By leveraging the available input and output mechanisms, shell scripts can effectively communicate with users and interact with external data sources, enhancing their functionality and versatility.

Here is an example of script:

```
#!/bin/bash
greeting="Welcome"
user=$(whoami)
day=$(date +%A)
echo "$greeting back $user! Today is $day, which the best
day of the week!"
echo "Your Bash shell version is: $BASH_VERSION. Enjoy!"
```

5.5 Statements

In shell scripting, statements are individual instructions or commands that are executed sequentially in a shell script. Shell scripts are composed of multiple statements, each performing a specific action or task. Here are some common types of statements used in shell scripting:

- **Command Execution:** Shell scripts execute various commands to perform tasks. Command execution statements are the most basic type of statement, where a command is written on a line and executed by the shell. For example:

```
echo "Hello, World!"
```

The above statement executes the echo command to print "Hello, World!" to the console.
- **Control Structures:** Control structures enable conditional execution and looping in shell scripts. They allow the script to make decisions based on conditions or repeat a block of code multiple times. Common control structures include:
if statements: Used for conditional branching based on a specified condition.
for loops: Used for iterating over a list of values or a range.
while and until loops: Used for executing a block of code repeatedly until a condition becomes true or false, respectively.
Control structures are typically written with specific syntax and use keywords such as `if`, `then`, `else`, `fi` (end of `if`), `for`, `do`, `done`, `while`, and `until`.
- **Comment Statements:** Comment statements are used to add explanatory or informative notes within the script. They are prefixed with a hash symbol (`#`) and are ignored by the shell when executing the script. Comments help improve script readability and provide documentation to others.

The `if` statement syntax is as follows:

```
if [condition]; then action(s); fi
```

For instance, the following script compare two number and display the greater of them:

```
#!/bin/bash
X=0
Y=10
If [$X -gt $Y]; then echo "greater number is $X"; fi
If [$X -le $Y]; then echo "greater number is $Y"; fi
```

It can be used also as follows:

```
if [ condition ]; then
    # Code block executed if the condition is true
else
    # Code block executed if the condition is false
fi
```

The `then` keyword marks the start of the code block that should be executed if the condition is true. The `else` keyword (optional) marks the start of the code block that should be executed if the condition is false. The `fi` keyword concludes the `if` statement.

Here is another example:

```
# Prompt user for their age
read -p "Enter your age: " age
# Check if age is greater than or equal to 18
if [ "$age" -ge 18 ]; then
```

```

    echo "You are an adult."
else
    echo "You are a minor."
fi

```

In the above script, the user's age is read from the input, and then the `if` statement checks whether the age is greater than or equal to 18. If it is, the script prints "You are an adult." Otherwise, it prints "You are a minor."

You can also use logical operators (`-a` for AND, `-o` for OR) and comparison operators (`-eq` for equal, `-ne` for not equal, `-lt` for less than, `-gt` for greater than, `-le` for less than or equal, `-ge` for greater than or equal, etc.) to form complex conditions within the `if` statement.

The `if` statement allows shell scripts to perform different actions based on specific conditions, providing flexibility and control over the execution flow of the script.

5.6 Loops

In shell scripting, loops are used to repeat a block of code multiple times until a specific condition is met. There are mainly three types of loops available: the `for` loop, the `while` loop, and the `until` loop.

- **for Loop:** The `for` loop iterates over a sequence of values, such as a list or a range of numbers. It executes a block of code for each value in the sequence. Here's the syntax of a `for` loop:

```

for variable in [list]; do
    # Code block to be executed
done

```

The `variable` takes on the values from the specified list in each iteration. The code block within the loop is executed for each value.

Example: Printing numbers from 1 to 5 using a `for` loop:

```

for i in 1 2 3 4 5; do
    echo $i
done

```

- **while Loop:** The `while` loop executes a block of code as long as a specified condition is true. The loop continues until the condition becomes false. Here is the syntax of a `while` loop:

```

while [ condition ]; do
    # Code block to be executed
done

```

The code block within the loop is executed repeatedly as long as the condition is true.

Example: Printing numbers from 1 to 5 using a `while` loop:

```

bash
counter=1
while [ $counter -le 5 ]; do

```

```

    echo $counter
    counter=$((counter+1))
done

```

- **until Loop:** The `until` loop is similar to the `while` loop, but it continues executing the code block until a specified condition becomes true. The loop keeps iterating until the condition is satisfied.

```

until [ condition ]; do
    # Code block to be executed
done

```

The code block within the loop is executed repeatedly until the condition evaluates to true.

Example: Printing numbers from 1 to 5 using an `until` loop:

```

counter=1
until [ $counter -gt 5 ]; do
    echo $counter
    counter=$((counter+1))
done

```

Loops are valuable for automating repetitive tasks, iterating over lists or ranges, and processing data based on specific conditions. They provide powerful control flow capabilities in shell scripts.

5.7 Arrays

In shell scripting, arrays are used to store multiple values in a single variable. Arrays allow you to group related data elements together for easier manipulation and processing. In most shells, arrays are indexed collections, starting from index 0. Here's how you can work with arrays in shell scripting:

- **Array Declaration:** To declare an array, you can use the following syntax:

```
array_name=(value1 value2 value3 ...)
```

Here is an example of declaring an array named `fruits`:

```
fruits=("apple" "banana" "orange" "mango")
```

- **Accessing Array Elements:** You can access individual elements of an array using the array name followed by the index in square brackets `[]`. For example:

```

echo ${fruits[0]}    # Access the first element
echo ${fruits[2]}    # Access the third element

```

- **Array Length:** To get the length or the number of elements in an array, you can use the `${#array_name[@]}` syntax. For example:

```
echo ${#fruits[@]}    # Get the length of the array
```

- **Modifying Array Elements:** You can assign new values to individual elements of an array by specifying the index within square brackets. For example:

```
fruits[1]="grape"     # Change the second element to
                    "grape"
```

- **Iterating over Array Elements:** You can iterate over the elements of an array using a `for` loop and the `${array_name[@]}` syntax. Here's an example:

```

for fruit in "${fruits[@]"; do
    echo $fruit

```



```
done
```

This will iterate over each element of the `fruits` array and print its value.

- **Deleting an Array:** To delete an entire array, you can use the `unset` command. For example:

```
unset fruits      # Delete the "fruits" array
```

Arrays in shell scripting provide a convenient way to store and manipulate collections of data. They can be used for various purposes, such as storing command-line arguments, processing lists of filenames, or managing sets of related values.

5.8 Functions

In shell scripting, functions are used to group a set of commands together as a reusable unit. They allow you to break down your script into smaller, modular components, making your code more organized, readable, and maintainable. Here's how you can define and use functions in shell scripting:

- **Function Definition:** To define a function, you can use the following syntax:

```
function_name() {  
    # Code block defining the function  
}
```

Here's an example of defining a function named `greet` that prints a greeting message:

```
greet() {  
    echo "Hello, $1!"  
}
```

To invoke or call a function, simply write its name followed by parentheses. If the function expects arguments, you can pass them within the parentheses. For example:

```
greet "Mohamed"
```

This will call the `greet` function with the argument "Mohamed". The function will then execute its code block.

- **Function Return:** In shell scripting, functions do not have a return value in the traditional sense. However, they can indicate success or failure using the exit status. By convention, an exit status of 0 indicates success, while any non-zero exit status indicates failure. You can use the `return` statement to explicitly set the exit status.

For example:

```
is_even() {  
    if (( $1 % 2 == 0 )); then  
        return 0      # Return success status  
    else  
        return 1     # Return failure status  
    fi  
}
```

In the above example, the `is_even` function checks if a number is even or not. It sets the exit status to 0 (success) if the number is even, and 1 (failure) if it's not.

- Local Variables: Inside a function, you can declare local variables using the local keyword. Local variables are only accessible within the function and do not affect variables with the same name outside the function. For example:

```
greet() {
    local name="John"
    echo "Hello, $name!"
}
```

In the above example, the name variable is local to the `greet` function and does not interfere with any variable named name outside the function.

Functions provide modularity and code reusability in shell scripting. They allow you to encapsulate a set of commands into a single unit, which can be invoked multiple times with different arguments. Functions can help make your shell scripts more structured and easier to maintain.

5.9 Conclusion

Shell scripting allows for the combination of multiple statements to create complex scripts that perform various tasks. By leveraging these different types of statements, shell scripts can automate processes, control program flow, handle input/output, and implement logic and decision-making within the script.

5.10 Exercises

- Assume we have variables location and work which store the city name and company name of the current user. Write an echo statement to output a greeting message including the user's username, his/her city name, and the company name.
- The script below will input two values from the user and sum them. What is wrong with the script? (there is nothing syntactically wrong). How would you correct it?


```
#!/bin/bash
read X Y
SUM=$((X+Y))
echo The sum of $X and $Y is $SUM
```
- If a person is 21 years or older, they are considered an adult, otherwise they are considered a child. Write a statement given the variable AGE storing the word adult or child in the variable STATUS.
- Write a script which will receive a list of parameters and will iterate through them and count the number of times the parameter is greater than 0. The script will output the number greater than 0, or an error message if no parameters are supplied.
- Write a function which receives a list of values as parameters and computes and outputs their sum.

References

- [1] Fox, R. (2021). *Linux with operating system concepts*. CRC Press.
- [2] Singh, R. (2014). An overview of android operating system and its security. *int. journal of Engineering Research and Applications*, 4(2), 519-521.
- [3] Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation (Vol. 2)*. Englewood Cliffs: Prentice Hall.
- [4] Sobell, M. G. (1997). *A practical guide to Linux*. Addison-Wesley Longman Publishing Co., Inc.
- [5] Koranne, S., & Koranne, S. (2011). GNU/Linux Operating System. *Handbook of Open Source Tools*, 3-34.
- [6] Kidwai, A., Arya, C., Singh, P., Diwakar, M., Singh, S., Sharma, K., & Kumar, N. (2021). A comparative study on shells in Linux: A review. *Materials Today: Proceedings*, 37, 2612-2616.
- [7] Yu, J., & Lu, X. (2023, January). Improvement and Application of Task Scheduling Algorithm for Embedded Real-Time Operating System. In *Proceedings of the World Conference on Intelligent and 3-D Technologies (WCI3DT 2022) Methods, Algorithms and Applications* (pp. 621-628). Singapore: Springer Nature Singapore.
- [8] Almarsoomi, F. A., & Alwan, I. A. (2022). A New Feature-Based Method for Similarity Measurement under the Linux Operating System. *International Journal of Interactive Mobile Technologies*, 16(18).
- [9] Hitchcock, K. (2022). Using Linux for the First Time. In *The Enterprise Linux Administrator: Journey to a New Linux Career* (pp. 123-158). Berkeley, CA: Apress.
- [10] Both, D., & Bulka, C. (2022). Using Linux–The Personal Case. In *Linux for Small Business Owners: Using Free and Open Source Software to Power Your Dreams* (pp. 1-13). Berkeley, CA: Apress.