

Operating Systems

by Asifa Iffat

Submission date: 25-Jul-2023 10:09PM (UTC+0100)

Submission ID: 210677636

File name: 44915_Asifa_Iffat_Operating_Systems_503754_1053557750.odt (2.67M)

Word count: 2495

Character count: 15289



COMPUTER ARCHITECTURE & OPERATING SYSTEMS

Asifa Iffat - student ID: 2226849



JULY 26, 2023

ACCA4022

University of Wales Trinity St David

Table of Contents

Operating Systems	1
Introduction and Evolution	2
Functions of Modern Operating Systems	3
CLI Commands	4
Example 1 of CLI commands.....	5
Example 2 of CLI commands	6
Batch scripts on Windows	7
Bash scripts on Linux.....	8
Continued Bash Scripts on Linux.....	9
Conclusion.....	10
Bibliography.....	11

Operating Systems: A Brief Introduction and Evolution

6

An operating system (OS) is a crucial software component that acts as an intermediary between hardware and software applications. It provides an environment for programs to run efficiently, manages system resources, and facilitates user interaction with the computer or device. Operating systems have evolved significantly since their inception, adapting to changing technological landscapes and user needs.

Operating systems are constantly evolving to meet the growing demands of modern computing. They form the backbone of the computing experience, enabling users to perform various tasks efficiently and facilitating the execution of complex applications across a wide range of devices.



11

Evolution of Operating Systems:

The concept of operating systems can be traced back to the 1950s when early computers were large, room-sized machines. These computers were operated using punch cards and required manual switching between programs. The introduction of batch processing systems in the 1960s allowed the execution of multiple jobs in sequence, but it still required a significant amount of manual intervention.

The 1970s witnessed the rise of time-sharing systems, which enabled multiple users to interact with a single computer simultaneously. Unix, developed in the early 1970s, was one of the pioneering operating systems that played a pivotal role in the OS evolution. With the advent of microprocessors, the 1980s saw the emergence of personal computers, leading to the popularity of operating systems like MS-DOS and Windows. In the 1990s and 2000s, advancements in graphical user interfaces (GUIs) and networking capabilities resulted in the widespread use of operating systems such as Windows, macOS, and various Linux distributions.

15

Types of Operating Systems:

Single-User, Single-Tasking OS: These operating systems can handle one task at a time and are suitable for small embedded systems or early personal computers (Stallings, 2018).

Single-User, Multi-Tasking OS: Most modern operating systems fall into this category. They allow users to run multiple applications simultaneously, sharing the CPU's time between tasks.

Multi-User OS: These OSs support multiple users accessing the system simultaneously, often found in servers and mainframes (Silberschatz, Galvin, & Gagne, 2018).

Real-Time OS: Designed for time-critical applications, these OSs ensure that tasks are executed within precise time constraints. Commonly used in industrial systems, robotics, and aerospace applications.

Key and Common Functions of Modern General Operating Systems

Error Handling: Operating systems employ error handling mechanisms to detect and recover from system errors and application crashes. They generate error messages and log information to assist in troubleshooting.

Printing Support: OSs provide printing capabilities, allowing users to send documents to printers and manage print jobs.

Backup and Restore: Modern operating systems include backup and restore functionalities to safeguard data against loss or corruption (Tanenbaum & Bos, 2014).

Task Scheduling: The OS scheduler determines the order and priority in which processes are executed, optimizing CPU utilization and responsiveness.

Inter-Process Communication (IPC): OSs facilitate communication between processes, enabling data sharing and coordination between applications.

Localization and Internationalization: Operating systems offer support for different languages, time zones, and cultural conventions to cater to users worldwide.

Accessibility Features: Modern OSs incorporate accessibility options for individuals with disabilities, such as screen readers, magnifiers, and voice recognition.

Virtual Memory Management: OSs use virtual memory to extend available RAM, allowing large programs to run efficiently even when physical memory is limited.

Software Installation and Management: Operating systems provide mechanisms to install, update, and uninstall software packages, ensuring application compatibility and security.

System Configuration: OSs allow users to customize system settings, such as display preferences, sound configurations, and network configurations.

System Monitoring and Performance Analysis: OSs monitor system performance, providing tools to analyse resource usage and identify bottlenecks.

Firewall and Network Security: Operating systems may include built-in firewalls to protect against unauthorized network access and potential threats.

System Recovery and Safe Mode: In the event of system instability or errors, modern OSs offer recovery options and safe mode for troubleshooting and repair.

Compatibility Layers: Some operating systems employ compatibility layers to run applications designed for different platforms, enhancing software availability.

Cloud Integration: Contemporary operating systems may integrate cloud services, enabling seamless synchronization of files and data across devices.

Encryption and Data Protection: OSs may provide encryption features to safeguard sensitive data stored on the system or transmitted over networks.

Speech Recognition and Voice Control: Some modern OSs incorporate speech recognition capabilities for hands-free control and interaction.

Gesture and Touch Support: Operating systems optimized for touch-based devices offer gesture support and multitouch capabilities.

CLI Commands:

CLI (Command-Line Interface) commands are text-based instructions given to a computer operating system or software application through the command-line interpreter. Unlike graphical user interfaces (GUI), where users interact with the system using windows, icons, and buttons, CLI commands require users to type specific commands directly into a command-line interface.

What Are CLI Commands:

CLI commands are specific instructions or statements that trigger actions or operations in the computer system. They typically consist of a command keyword followed by optional parameters and arguments. The command-line interpreter then processes these commands and executes the corresponding actions.

Advantages of CLI Commands:

Efficiency: CLI commands can be faster to use than navigating through GUI menus, especially for experienced users who know the commands well.

Automation: CLI commands are ideal for automating tasks and creating scripts to perform repetitive operations.

Flexibility: Command-line interfaces often provide more options and configurations than their graphical counterparts.

Remote Access: CLI commands are commonly used for remote administration of systems and servers.

Resource-Efficient: CLI environments use fewer system resources than GUI environments, making them useful for low-resource systems or remote access with slow connections.

type (Windows) / **cat** (Linux/macOS): Display file contents

find (Windows) / **grep** (Linux/macOS): Search for text in files

Archiving and Compression:

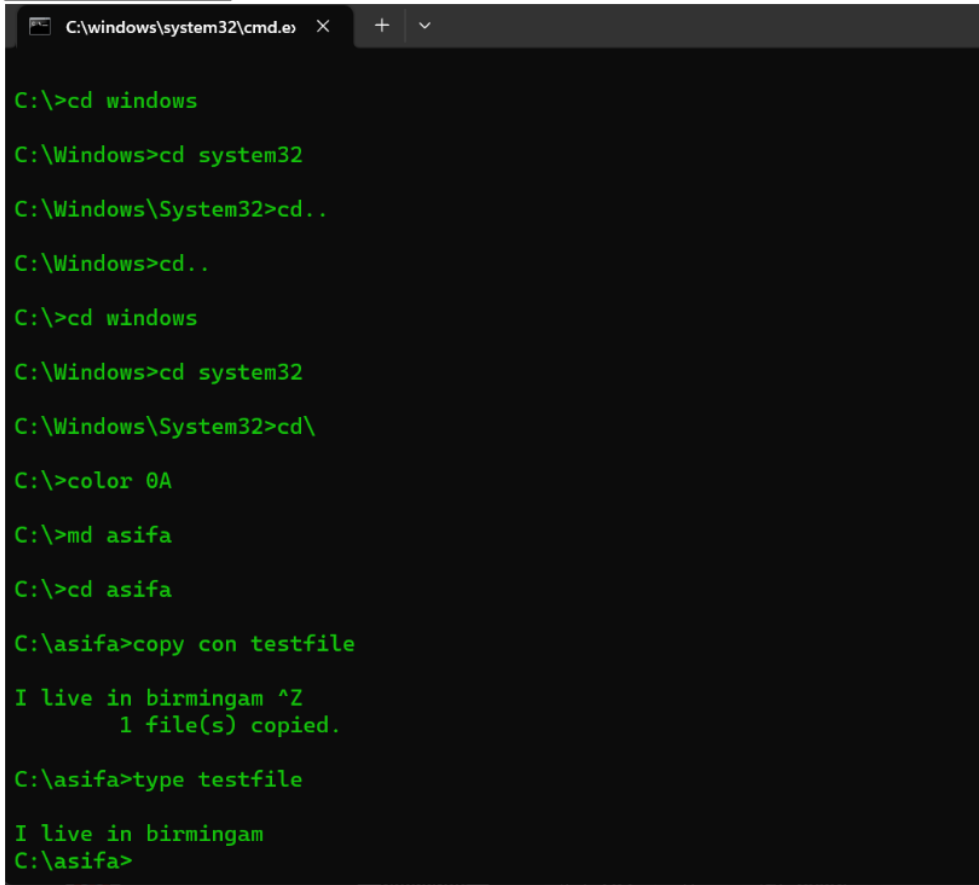
compress (Windows) / **zip** (Linux/macOS): Create a compressed archive

expand (Windows) / **unzip** (Linux/macOS): Extract files from a compressed archive

These examples are just a glimpse of the many CLI commands available, and each operating system and software may have its own set of unique commands and syntax.

CLI commands are powerful tools for system administrators, developers, and advanced users, allowing them to control their systems efficiently and automate tasks with precision. Learning and mastering CLI commands can significantly enhance one's ability to work with computer systems effectively.

Example 1



```
C:\>cd windows
C:\Windows>cd system32
C:\Windows\System32>cd..
C:\Windows>cd..
C:\>cd windows
C:\Windows>cd system32
C:\Windows\System32>cd\
C:\>color 0A
C:\>md asifa
C:\>cd asifa
C:\asifa>copy con testfile
I live in birmingham ^Z
    1 file(s) copied.
C:\asifa>type testfile
I live in birmingham
C:\asifa>
```

"cd windows" → command used to change to directory.

"cd system32" → command used to change directory. Then I returned to the previous directory using the command "cd..".

"color 0A" → command used to alter font colour.

"md Asifa" → command used to create a new directory.

"cd Asifa" → command used to enter in folder. Then you can write anything on this file so I wrote "I live in birmingham"

"type testfile" → command used to view the input of your file.

Example 2:

```
C:\windows\system32\cmd.e) X + v

C:\>cd asifa1

C:\asifa1>copy con testfile
THIS IS TEST FILE^Z
        1 file(s) copied.

C:\asifa1>dir
Volume in drive C is Windows
Volume Serial Number is FC25-B18D

Directory of C:\asifa1

22/07/2023  15:26    <DIR>          .
22/07/2023  15:26                  17 testfile
                1 File(s)                17 bytes
                1 Dir(s) 181,676,605,440 bytes free

C:\asifa1>ren testfile importantfile

C:\asifa1>dir
Volume in drive C is Windows
Volume Serial Number is FC25-B18D

Directory of C:\asifa1

22/07/2023  15:29    <DIR>          .
22/07/2023  15:26                  17 importantfile
                1 File(s)                17 bytes
                1 Dir(s) 181,676,548,096 bytes free
```

To begin, I entered the directory using the command “cd asifa1”. Then, I created a file using the command “copy con filename”. To check if the file is successfully created I used the command “dir”. To rename the file I used the command “ren previousfilename newfilename” and to confirm this I once again used the command “dir” which displayed the new file details such as serial number which is the same as the one before we renamed the file, thus it must be the same file.

Using Batch Scripts in Windows

Batch scripts in Windows are essentially text files containing a series of command-line instructions that are executed in sequence. They allow users to automate repetitive tasks and perform various operations on the system. Batch scripts have the file extension ".bat" or ".cmd" and are executed by the Command Prompt or Windows PowerShell. Below are two examples of batch scripts in Windows:

Key Features of Batch Scripts:

Automating Tasks: Batch scripts are designed to automate repetitive tasks on Windows systems. They can be used to perform tasks like file management, system configuration, running multiple programs sequentially, and more.

Command Execution: Similar to Bash scripts, batch scripts can execute a sequence of command-line instructions. These commands can include system commands, utilities, or other executable files.

Variables: Batch scripts can use variables to store and manipulate data temporarily. Variables are denoted with the "%" symbol (e.g., %variable%), and they allow batch scripts to perform conditional operations and parameter substitution.

Control Structures: Batch scripts support control structures like loops (e.g., FOR, WHILE) and conditional statements (e.g., IF, ELSE) to control the flow of the script based on conditions.

User Interaction: Batch scripts can prompt the user for input and use that input to guide the script's execution dynamically.

Creating a Simple Script:

```
1 @echo off
2 echo display year month week and date please
3
4
5 set/p calander= enter the year name
6
7 set/p calendert= enter the month name
8
9 set/p calender= enter the date
10
11 echo %calender%
12
13 pause
```

This batch script creates a backup of all files in the "Documents" folder and places them in a new directory called "Backup" on the C:\ drive. The @echo off command disables the display of commands in the output, and echo is used to provide status messages. Finally, pause keeps the Command Prompt window open after the backup is

complete. To begin, I used the command "@echo off". I used the command "echo display year month week and date please" and finally to set the date I used the following variable "set/p calender=".

Displaying System Information:

```
2  echo OFF
3
4  ECHO You want coffee?
5
6  ECHO Enter Y for yes
7
8  ECHO Enter N for no
9
10 CHOICE /c YN /m "Yes or No"
11
12 echo thankz
13
14 pause
```

This batch script collects system information using the systeminfo command and redirects the output to a text file named "system_info.txt" on the desktop. The systeminfo command provides details like OS version, memory, processor information, etc. The pause command is used to keep the Command Prompt window open after executing the script.

Using CLI Commands on Linux

Linux operating systems offer powerful command-line interfaces (CLI) that provide users with extensive control over the system. The Terminal application is commonly used to access the CLI on Linux distributions.



Linux

Using Bash Scripts on Linux

Bash scripts in Linux are text files containing a series of commands written in the Bash shell scripting language. They allow users to automate tasks, perform system configurations, and execute complex operations. Bash is the default shell for most Linux distributions and provides powerful scripting capabilities. Let's explore an example of Bash script on Linux:

```
onworks@onworks-Standard-PC-l440FX-PIIX-1996: ~/Desktop
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ nano math.sh
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ chmod math.sh
chmod: missing operand after 'math.sh'
Try 'chmod --help' for more information.
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ bash math.sh
Enter the first number:
2
Enter the second number:
3
The sum is: 6
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$
```

BRIEF:

Start off by creating a file with the following command "nano insert file name.sh". Once the file is named I wrote the following onto it:

```
5 echo "Enter the first number"
read num1
echo "Enter the second number"
read num2
sum=$((num1 * num2))
echo "The sum is: $sum"
```

Then I used the bash command to test out if it working correctly so I inserted the following “bash file name. sh”. Then I inserted 2 random numbers as per stated which gave me a correct statement in the end.

```
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ nano math2.sh
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ bash math2.sh
Enter the first number:
12
Enter the second number;
6
The sum is: 6
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$
```

Similarly, for this second example I used the same commands except when I was typing in the file, I replaced “sum=\$((num1 * num2))” with “sum=\$((num1 – num2))” to solve a different mathematical problem.

Linux CLI Command:

```
onworks@onworks-Standard-PC-l440FX-PIIX-1996: ~/Desktop
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ nano test.txt
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ cat test.txt
This file has no purpose as it is a test file.
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ ls -l
total 4
-rw-r--r-- 1 onworks onworks 48 Jul 23 17:20 test.txt
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ rm test.txt
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ ls -l
total 0
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$
```

BRIEF:

To begin, you must create a file so I created one using the following command “nano filename.txt”. Then insert a piece of text on the file once its created which the system can display after, so on the file I wrote “This file has no purpose as it is a test file”. Then we need to test if the file works so in order to do this type the following “cat filename.txt”. This will display whatever text you wrote on your file. Then using the command “ls -l” which stated the details of file. Then I removed the file by using the following command “rm filename.txt”. To test if this command was successful I used the “ls -l” command to state the details of the file which stated “total 0” so we know that the file has been removed.



```
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ nano testfile2.txt
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ cd testfile2.txt
bash: cd: testfile2.txt: Not a directory
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ pwd
/home/onworks/Desktop
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ ls
testfile2.txt
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$
```

I used multiple commands in this second example such as “nano filename” to create a file and “cd” to change directory. However, you will notice that the “cd” did not do anything. This is because my file is not a directory thus it will not change. Another command I used is “pwd” which prints the working directory and “ls” command to list the components of the directory.

Conclusion:

In conclusion, we have explored the fundamental concepts and evolution of operating systems, understanding their crucial role in managing hardware resources, providing an interface for user interaction, and supporting application execution. Operating systems have evolved significantly over the years, from early batch processing systems to the emergence of time-sharing, personal computing, and modern multi-user, multi-tasking environments. Today, operating systems like Windows, macOS, Linux, and others cater to diverse user needs, from desktop computing to servers and embedded systems.

Furthermore, we delved into the key and common functions of modern general operating systems. These functions include process and memory management, file system management, device handling, user interface, security measures, networking capabilities, virtualization, power management, and system utilities. Understanding these functions gives us insight into the underlying mechanisms that allow our devices to perform efficiently and provide a seamless user experience.

Additionally, we explored the command-line interfaces (CLI) of Windows, macOS, and Linux. The CLI offers direct control over the operating system and is favored by power users and administrators for its ability to automate tasks and perform complex operations efficiently. We learned how to navigate the file system, list directory contents, and use batch scripts in both Windows and Linux. These scripts empower users to automate tasks, improving productivity and reducing manual intervention.

Overall, this exploration into operating systems and their functionalities has provided valuable insights into the foundations of modern computing. Understanding how operating systems work enables users to make informed decisions, troubleshoot issues, and utilize the full potential of their devices. Additionally, command-line interfaces and scripting open up a world of possibilities for automating tasks and streamlining workflows. As technology continues to evolve, the knowledge gained from this exploration will remain relevant and valuable in the dynamic landscape of computing and beyond.

Bibliography:

- ¹ Tanenbaum, A. S., & Bos, H. (2014). *Modern Operating Systems* (4th ed.). Prentice Hall.
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
- Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.
- ⁸ The Linux Foundation. (n.d.). Introduction to Linux. Retrieved from <https://training.linuxfoundation.org/training/introduction-to-linux/>
- ¹⁴ Microsoft. (n.d.). Windows: How it works. Retrieved from <https://www.microsoft.com/en-us/windows/for-business/how-windows-works>
- Apple. (n.d.). macOS. Retrieved from <https://www.apple.com/macOS/>
- ² Android Developers. (n.d.). Android OS. Retrieved from <https://developer.android.com/studio/intro>
- ⁴ Red Hat. (n.d.). What is Red Hat Enterprise Linux? Retrieved from <https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux>
- Canonical Ltd. (n.d.). Ubuntu. Retrieved from <https://ubuntu.com/>
- ¹⁰ The FreeBSD Project. (n.d.). FreeBSD. Retrieved from <https://www.freebsd.org/>
- ¹³ Microsoft. (n.d.). Command Prompt: Command-line reference. Retrieved from <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/command-prompt>
- ¹⁶ Linux.org. (n.d.). Linux Man Pages. Retrieved from <https://www.linux.org/docs/man-pages/>
- Ubuntu. (n.d.). Terminal. Retrieved from <https://help.ubuntu.com/community/UsingTheTerminal>
- Microsoft. (n.d.). Batch files - Windows CMD. Retrieved from <https://ss64.com/nt/>
- ⁹ Techopedia. (n.d.). What is Batch File? - Definition from Techopedia. Retrieved from <https://www.techopedia.com/definition/3370/batch-file>
- ³ Linux Documentation Project. (n.d.). Advanced Bash-Scripting Guide. Retrieved from <http://tldp.org/LDP/abs/html/>
- GNU Project. (n.d.). Bash Reference Manual. Retrieved from <https://www.gnu.org/software/bash/manual/bash.html>

Operating Systems

ORIGINALITY REPORT

11%

SIMILARITY INDEX

5%

INTERNET SOURCES

2%

PUBLICATIONS

10%

STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Washington University of Science and Technology Student Paper	1%
2	Submitted to Colorado State University, Global Campus Student Paper	1%
3	Submitted to University of Wales, Lampeter Student Paper	1%
4	Submitted to De Montfort University Student Paper	1%
5	Submitted to Pathfinder Enterprises Student Paper	1%
6	Submitted to University of Wales Institute, Cardiff Student Paper	1%
7	Submitted to HCUC Student Paper	1%
8	Submitted to Middle East College of Information Technology Student Paper	1%

9	www.powtoon.com Internet Source	1 %
10	Submitted to Kaplan University Student Paper	1 %
11	Submitted to Victoria University Student Paper	1 %
12	Submitted to Swansea College Student Paper	<1 %
13	s3-eu-west-1.amazonaws.com Internet Source	<1 %
14	journals.humankinetics.com Internet Source	<1 %
15	www.coursehero.com Internet Source	<1 %
16	www.frontiersin.org Internet Source	<1 %

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off