

Assessment 2

by Alexei Nica

Submission date: 26-Jul-2023 03:23PM (UTC+0100)

Submission ID: 210708705

File name: 44825_Alexei_Nica_Assessment_2_503754_1173416017.docx (2.4M)

Word count: 3314

Character count: 18066



OPERATING SYSTEMS

Module: Computer Architecture & Operating Systems

Module code: ACCA4022

Lecturer: Md Shantanu Islam

Student name: Alexei Nica

Student ID number: 2232919

Submission deadline: 26 July 2023

Table of Contents

Introduction	page 2
1. Operating Systems and their key functions.....	page 3
1.1. What is an Operating System, brief introduction and evolution?	page 3
1.2. Types of Operating System.....	page 4
1.3. Introduce and analyse the key and common functions of modern general Operating Systems.....	page 4
2. Use of CLI commands in a Windows ..	page 5
2.1. Document the use of CLI commands using a Windows	page 5
2.2. Give at least two examples with brief explanation and screenshots.....	page 6
3. Use of CLI commands in a Linux OS.....	page 9
3.1. Document the use of CLI commands using a Linux OS.	page 9
3.2. Give at least two examples with brief explanation and screenshots.....	page 9
4. Use of Batch scripts in a Windows.....	page 12
4.1. Document the use of batch scripts using a Windows	page 12
4.2. Give at least two examples with brief explanation and screenshots.....	page 12
5. Use of Bash scripts in a Linux OS.....	page 16
5.1. Document the use of bash scripts using a Linux OS.....	page 16
5.2. Give at least two examples with brief explanation and screenshots.....	page 16
Conclusion	page 19
Reference list	page 20

Introduction:

This assessment is designed to deepen our understanding of Operating Systems (OS), a critical software that serves as an intermediary between users and computer hardware. I start with a comprehensive analysis of OS, its evolution, and its various types. Further, I delve into the fundamental functionalities of modern general OS, elucidating their critical roles.

The latter part of the assessment will provide practical exposure to the command-line interface (CLI) commands, an integral part of OS. The CLI, often considered an OS's backbone, is a user interface that allows users to interact with the OS using textual commands. The command-line interface, available in OS like Windows and Linux, will be examined. Proficiency in utilizing CLI commands is an essential skill for IT professionals, which forms an integral part of this assessment.

Following the CLI, we will explore batch scripting in Windows. Batch scripts are text files containing a series of commands executed sequentially. This knowledge is vital for automating repetitive tasks, significantly increasing productivity.

Finally, we will learn about bash scripting in Linux OS, a powerful tool that combines the CLI commands' efficiency and the programming features like variable declaration and looping.

The inclusion of real examples and screenshots will serve to enhance understanding and make theoretical knowledge practically applicable. The references cited throughout will ensure that information is validated and can lead to further reading for curious learners.

1

1. Operating Systems and their key functions

1.1. What is an Operating System, brief introduction and evolution?

1.2. Types of Operating Systems

1.3. Introduce and analyze the key and common functions of modern general Operating Systems.

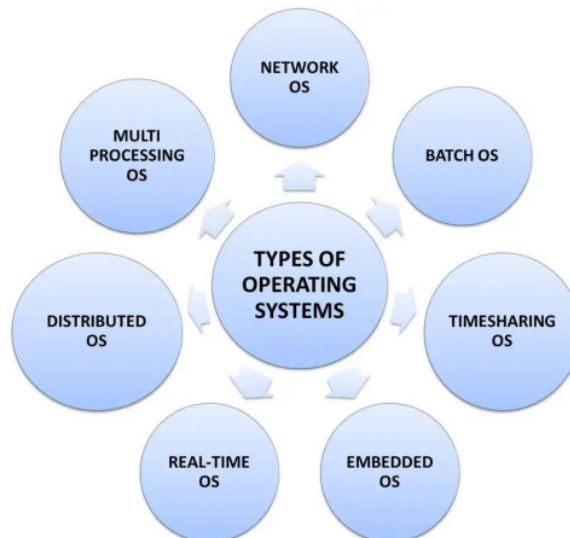
1

An operating system (OS) is a vital piece of software that interfaces between the user and the hardware in a computing system. It controls the allocation and utilization of hardware resources such as the CPU, memory, disk I/O, and peripheral devices. As a systems software, the OS ensures the seamless operation of other software, typically applications, by providing services like scheduling tasks, handling system calls, and managing files and main memory (Stallings & Brown, 2018).

The evolution of operating systems spans over six decades. Early computers did not have an OS, and programmers interacted directly with the hardware. This era was followed by the advent of batch-processing systems in the 1950s. Then, in the 1960s, multiprogramming and time-sharing systems became prominent, setting the stage for more complex OSs. The Personal Computer (PC) revolution of the 1980s introduced user-friendly graphical interfaces, and finally, the 2000s saw the birth of modern, feature-rich, and internet-capable OSs (Silberschatz, Galvin, & Gagne, 2018).

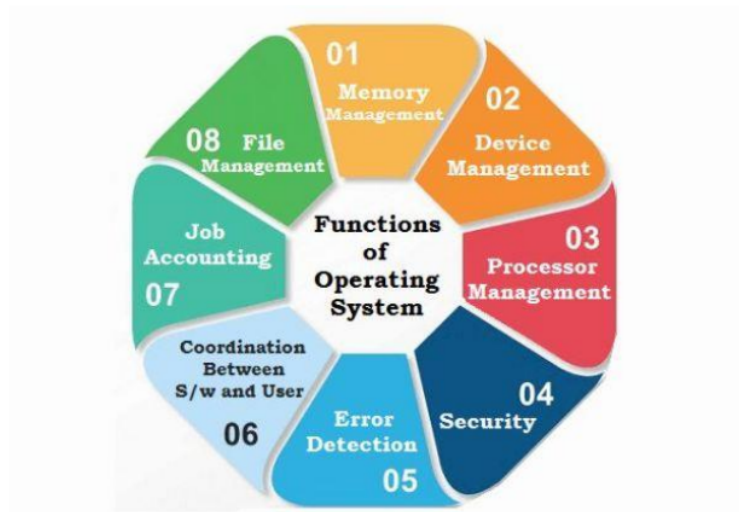
12

Operating systems can be classified into various types:



- **Batch Operating Systems:** These systems do not interact with the user directly. Similar types of processes are grouped for execution as a batch.
- **Time-Sharing Operating Systems:** These systems are intended to share system resources among multiple users at a time.
- **Distributed Operating Systems:** These manage a group of distinct computers and make them appear as a single computer.
- **Network Operating Systems:** These are designed to manage and coordinate networked computers.
- **Real-Time Operating Systems:** These are used when there are rigid time requirements on the operation of a processor or the flow of data.

Modern operating systems like Windows, macOS, and Linux, while distinct in their specific functionalities and user interfaces, share some common, key functions:



Process Management: The OS handles all the processes (programs in execution) in the system. It is responsible for process scheduling, creation, termination, and interprocess communication. This is essential for multitasking and optimizing the system's speed and efficiency (Stallings, 2018).

Memory Management: Memory management involves managing the computer's primary memory or RAM. The OS keeps track of each byte in the memory and decides which processes will get memory, when they will receive it, and how much they are allowed. This aids in memory protection, allocation, and deallocation (Tanenbaum & Bos, 2014).

1 File System Management: The OS provides a way for programs and users to create, write, read, delete, and manage files. It controls access to files and manages the organization of files on the storage device.

11 Device Management: The OS manages device communication via respective drivers. It serves as an intermediary between your hardware components and your software applications, ensuring smooth interaction (Arpaci-Dusseau, 2014).

Security and Access Control: The OS secures the system through user authentications and access control policies to prevent unauthorized access (Bovet & Cesati, 2005).

7 User Interface: Operating systems offer either a Command Line Interface (CLI) or Graphical User Interface (GUI) or both. The user interface allows humans to interact with the computer system more conveniently.

While these functions are found in most operating systems, each OS might implement them differently, considering the specific system's objectives and hardware.

In conclusion, an OS is a fundamental component of any computer system. From managing hardware to providing an interface for users, its roles are numerous and crucial. In its ongoing evolution, the OS continues to be a critical area of research and development, with future trends focusing on aspects like cloud compatibility, cybersecurity, and artificial intelligence integration.

1 2. Use of CLI commands in a Windows

- **Document the use of CLI commands using a Windows**
- **Give at least two examples with brief explanation and show screenshots.**

The Command-Line Interface (CLI) is a powerful tool in the Windows operating system, offering users an additional, text-based mode to interact with their systems. Despite the prevalence of graphical interfaces, CLI remains an essential component in modern computing (Frish, 2016).

CLI, typically accessed via Windows Command Prompt, provides several commands for managing files, network configurations, and system information. For example, the **ipconfig** command helps users display or manage network configurations, providing crucial data when troubleshooting network issues (Stanek, 2014). The **dir** command, on the other hand, enables users to list the contents of a directory, offering a quick way to navigate file structures.

Furthermore, the **net** command set provides a range of network and system administrative functions, such as **net user** for managing user accounts. (Sosinsky, 2011).

In my first example, I created a directory and created a file in my new directory.

In the first step, I opened Command Prompt from the search bar. By default, it set me in “C:\Users\nadyy>” directory. I use “dir” command to see my directories in that location and choose the Documents directory to create my new map. It’s shown in the screenshot below:

```
C:\Users\nadyy>dir
Volume in drive C is Acer
Volume Serial Number is 78D0-F09E

Directory of C:\Users\nadyy

12/07/2023  10:57  <DIR>          .
16/01/2023  02:07  <DIR>          ..
12/07/2023  10:57                0 ++
16/01/2023  08:19  <DIR>          Contacts
24/07/2023  10:20  <DIR>          Documents
26/07/2023  11:31  <DIR>          Downloads
16/01/2023  08:19  <DIR>          Favorites
16/01/2023  08:19  <DIR>          Links
16/01/2023  08:19  <DIR>          Music
23/07/2023  12:33  <DIR>          OneDrive
16/01/2023  08:19  <DIR>          Saved Games
16/01/2023  08:19  <DIR>          Searches
22/01/2023  11:01  <DIR>          Videos
               1 File(s)              0 bytes
               12 Dir(s)  215,509,139,456 bytes free

C:\Users\nadyy>cd Documents
C:\Users\nadyy\Documents>
```

Screenshot 1

¹ To create a new directory I used the command “mkdir” and the name of my new directory:

```
C:\Users\nadyy\Documents>mkdir TestDir

C:\Users\nadyy\Documents>dir
Volume in drive C is Acer
Volume Serial Number is 78D0-F09E

Directory of C:\Users\nadyy\Documents

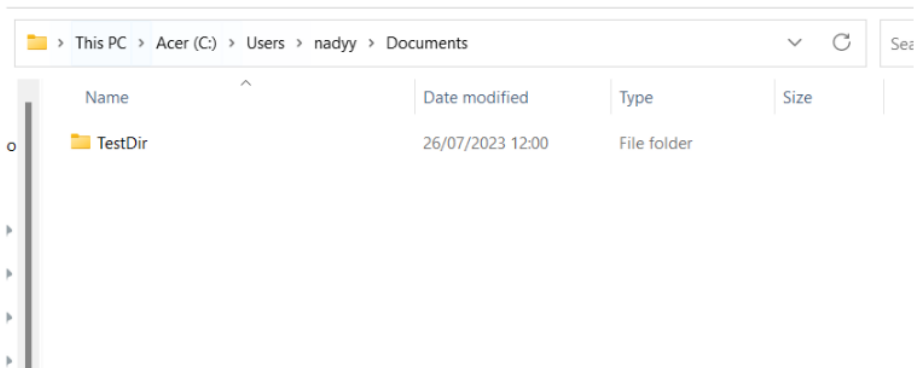
26/07/2023  12:00  <DIR>          .
12/07/2023  10:57  <DIR>          ..
26/07/2023  12:00  <DIR>          TestDir
               0 File(s)              0 bytes
               3 Dir(s)  215,483,301,888 bytes free

C:\Users\nadyy\Documents>
```

Screenshot 2

I named my directory TestDir and verified if it was created.

As well, I checked my map in Windows:



To create a file in my new directory I use the command echo and name of my file with extension. In next screenshot, it related to how I created and verified my new file.

```
C:\WINDOWS\system32\cmd. x + v

C:\Users\nady\Documents\TestDir>echo > Test.txt

C:\Users\nady\Documents\TestDir>dir
Volume in drive C is Acer
Volume Serial Number is 78D0-F09E

Directory of C:\Users\nady\Documents\TestDir

26/07/2023  12:18    <DIR>          .
26/07/2023  12:00    <DIR>          ..
26/07/2023  12:18                13 Test.txt
               1 File(s)                13 bytes
               2 Dir(s)  215,469,907,968 bytes free

C:\Users\nady\Documents\TestDir>
```

In my second example, I verified my history of recently executed commands, check the tasks that are opened, and deleted the created file.

```

C:\Users\naddy\Documents\TestDir>doskey/history
dir
Documents
cd Documentscls
cls
dir
cd Documents
mkdir TestDir
dir
cls
cd TestDir
echo Test.txt
dir
echo Test.txt
echo > Test.txt
dir
cls
echo > Test.txt
dir
copy Test.txt
doskey/history

```

Because the list with opened tasks is too large, I made a screenshot for a lot of them:

```

C:\Users\naddy\Documents\TestDir>tasklist

```

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0	Services	0	8 K
System	4	Services	0	136 K
Registry	140	Services	0	30,704 K
smss.exe	544	Services	0	336 K
csrss.exe	860	Services	0	2,160 K
wininit.exe	948	Services	0	804 K
csrss.exe	968	Console	1	3,340 K
services.exe	1020	Services	0	5,820 K
lsass.exe	604	Services	0	17,556 K
winlogon.exe	696	Console	1	4,816 K
svchost.exe	1120	Services	0	36,504 K
fontdrvhost.exe	1140	Console	1	5,832 K
fontdrvhost.exe	1156	Services	0	1,328 K
WUDFHost.exe	1212	Services	0	1,392 K
svchost.exe	1280	Services	0	18,604 K
svchost.exe	1352	Services	0	3,376 K
dwm.exe	1420	Console	1	215,488 K

To delete a file, I used the command `del` and the name of my file:

```

C:\Users\naddy\Documents\TestDir>del Test.txt

C:\Users\naddy\Documents\TestDir>dir
Volume in drive C is Acer
Volume Serial Number is 78D0-F09E

Directory of C:\Users\naddy\Documents\TestDir

26/07/2023  12:28    <DIR>        .
26/07/2023  12:00    <DIR>        ..
               0 File(s)                0 bytes
               2 Dir(s)  215,457,820,672 bytes free

C:\Users\naddy\Documents\TestDir>

```

In summary, understanding and using CLI commands in Windows is integral for executing many advanced computing tasks. It offers a level of control that's typically unmatched by graphical user interfaces. As a result, professionals across various IT fields often rely on CLI for system management, troubleshooting, and automation purposes.

3. Use of CLI commands in a Linux OS.

- Document the use of CLI commands using a Linux OS.
- Give at least two examples with brief explanation and show screenshots.

Command Line Interface (CLI) commands are vital for managing and navigating within a Linux Operating System. They provide a way to perform operations without the need for a graphical interface, which can be especially important for remote access, server management, or automation (Shotts, 2019).

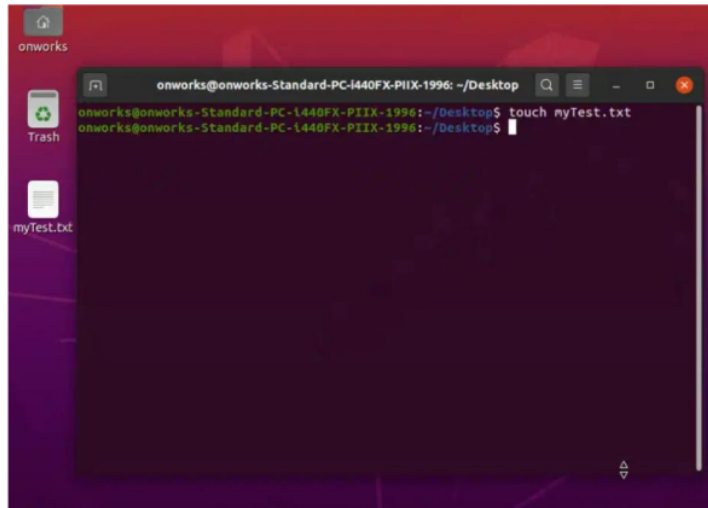
The 'ls' command, for instance, is one of the most fundamental, as it allows users to list the contents of a directory, offering a simple yet powerful tool to explore the filesystem (Nemeth, et al., 2011).

Another frequently used command is 'cd', which is used to change the current working directory. By using 'cd' along with a path, users can navigate through the system's directory hierarchy seamlessly (Barrett, 2020).

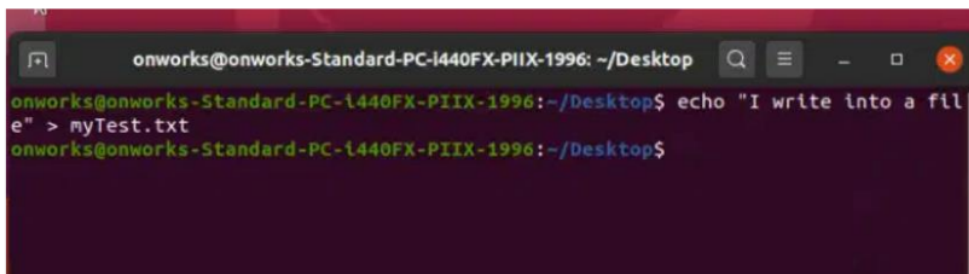
Ultimately, these commands illustrate the power and flexibility inherent in the Linux command line, allowing users to navigate, view, and manipulate their files efficiently. Their utility underscores the importance of CLI proficiency for those using Linux.

In the next examples, I used some commands to demonstrate practical knowledge of CLI in Linux.

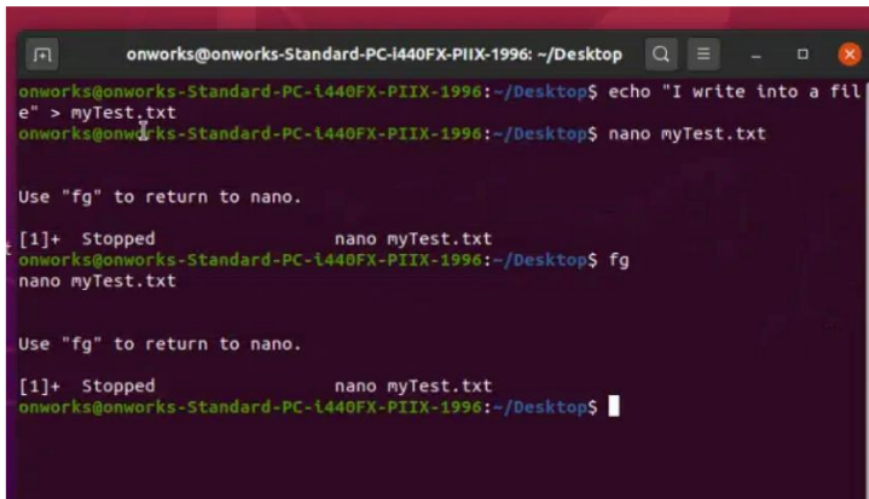
First, I created a .txt file and name it (See screenshot):

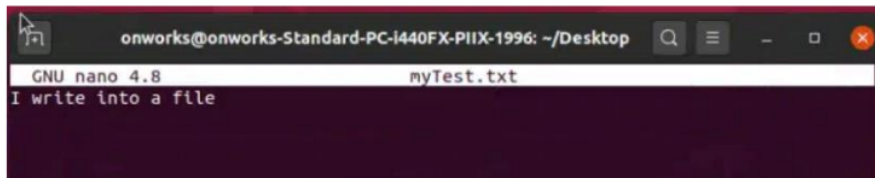


Next, I wrote a text in my created document.



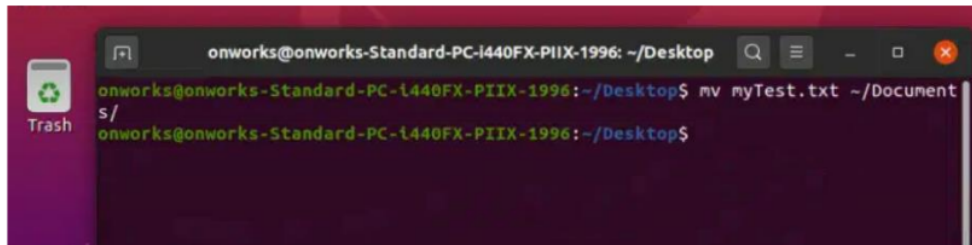
In the next screenshots, is seen how I open my document and return to Terminal.





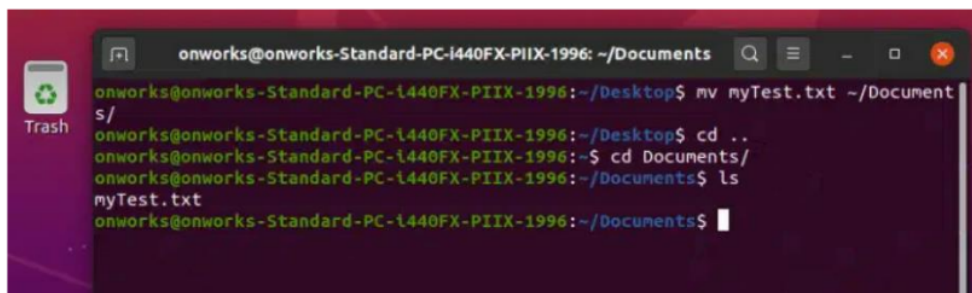
```
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop
GNU nano 4.8 myTest.txt
I write into a file
```

After, I moved my file from Desktop to Document and



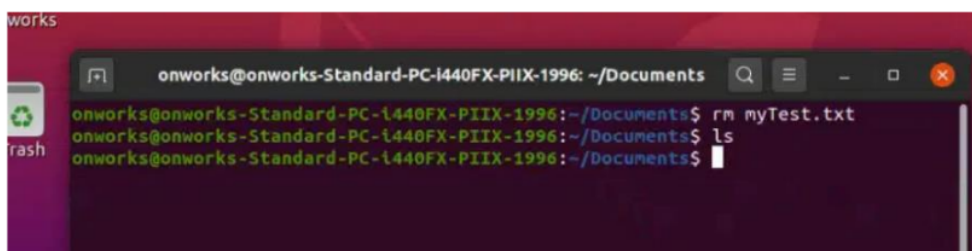
```
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Desktop
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ mv myTest.txt ~/Documents/
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$
```

I moved back from my directory using the command cd.., entered the Documents directory, and verified if my files is there:



```
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Documents
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ mv myTest.txt ~/Documents/
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ cd ..
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~$ cd Documents/
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Documents$ ls
myTest.txt
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Documents$
```

Finally, I removed my file:



```
onworks@onworks-Standard-PC-i440FX-PIIX-1996: ~/Documents
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Documents$ rm myTest.txt
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Documents$ ls
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Documents$
```

4. ¹Use of Batch scripts in a Windows

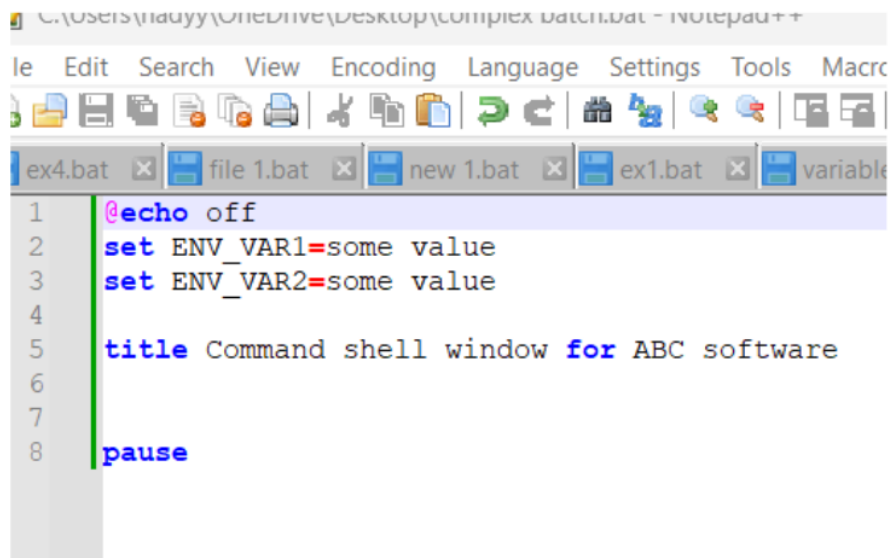
- Document the use of batch scripts using a Windows
- Give at least two examples with brief explanation and show screenshots.

Batch scripting in a Windows environment is a crucial tool for automating repetitive tasks, thus increasing productivity, and reducing potential errors. Scripts are essentially a series of command line instructions that are processed sequentially by the Command Prompt shell (Flylib, n.d.).

One significant application of batch scripts is in file operations. For instance, the "rename" command allows multiple files to be renamed simultaneously, saving the user time and effort in performing manual renames. It achieves this by using wildcards to select a set of files and rename them according to a specified format (Microsoft, 2022).

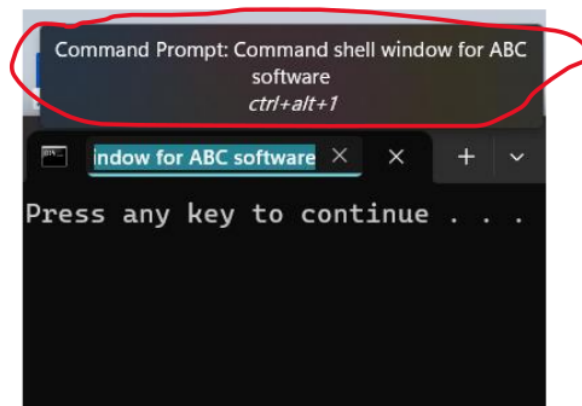
Batch scripts are powerful because of their adaptability and simplicity. With a basic understanding of command line syntax and the correct usage of commands, complex tasks can be automated with relatively simple scripts.

In my first example, my batch changes the title of the command prompt window.



```
1 @echo off
2 set ENV_VAR1=some value
3 set ENV_VAR2=some value
4
5 title Command shell window for ABC software
6
7
8 pause
```

And the result:



@echo off: This command is used to prevent the commands in the script from being displayed as they're executed. The @ symbol before the command prevents the command itself from being displayed.

set ENV_VAR1=some value: This line sets an environment variable named ENV_VAR1 to "some value". Environment variables are variables that are available system-wide and can be used to configure the system's behavior.

set ENV_VAR2=some value: This line does the same as the previous line but sets a different environment variable (ENV_VAR2).

title Command shell window for ABC software: This command changes the title of the command prompt window to "Command shell window for ABC software".

pause: This command pauses the execution of the script and waits for the user to press any key before continuing. It's often used at the end of scripts to prevent the command prompt window from closing immediately after the script is run, allowing the user to read any output.

For the second example, I used the last example from the in-class list:

```
C:\Users\naddy\OneDrive\Desktop\new 1.bat - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
new 1.bat x
1 @echo off
2 :TryAgain
3 set /p height=Enter the height of the pyramid:
4 setlocal enabledelayedexpansion
5 set "spaces= "
6 set "asterisk=*"
7 rem Calculate the number of spaces required for the first row
8 set /a spaces_num=%height%-1
9 rem Loop through each row of the pyramid
10 for /l %i in (1, 1, %height%) do (
11     rem Print the required number of spaces
12     for /l %j in (1, 1, !spaces_num!) do (
13         echo|set /p=!spaces!
14     )
15     rem Print the required number of asterisks
16     for /l %k in (1, 1, %i) do (
17         echo|set /p=!asterisk!
18     )
19     rem Move to the next line
20     echo.
21     rem Decrement the number of spaces for the next row
22     set /a spaces_num-=1
23 )
24 endlocal
25 goto :TryAgain
```

I slightly modified the batch, by adding :TryAgain function, to repeat the process until I finish manually the batch. And the result is:

```
C:\WINDOWS\system32\cmd. x + v
Enter the height of the pyramid: 3
*
* *
* * *
Enter the height of the pyramid: 4
*
* *
* * *
* * * *
Enter the height of the pyramid: 6
*
* *
* * *
* * * *
* * * * *
* * * * *
Enter the height of the pyramid: 1
*
Enter the height of the pyramid:
```

set /p height=Enter the height of the pyramid:: This command prompts the user to input the height of the pyramid, which is stored in the height variable.

setlocal enabledelayedexpansion: This command enables the use of delayed environment variable expansion, which allows you to use the !var! syntax to access the current value of a variable inside a loop.

set "spaces= ", set "asterisk=*": These lines define two variables, spaces and asterisk, which contain a space and an asterisk, respectively.

set /a spaces_num=%height%-1: This line calculates the number of spaces that should be printed before the asterisks on the first row of the pyramid.

for /l %%i in (1, 1, %height%) do (: This command starts a loop that iterates over each row of the pyramid.

The first inner loop (**for /l %%j in (1, 1, !spaces_num!) do ()**) is used to print the required number of spaces on each row.

The second inner loop (**for /l %%k in (1, 1, %%i) do ()**) is used to print the required number of asterisks on each row.

set /a spaces_num-=1: This line decreases the number of spaces for the next row.

endlocal: This command ends the scope of setlocal, making the changes to environment variables inside the setlocal block local to that block.

goto :TryAgain: This command jumps back to the **:TryAgain** label, repeating the process until I manually terminate the script.

This batch script creates a user-specified pyramid pattern of asterisks using nested loops. It calculates spaces before asterisks for pyramid alignment and repeats the process until manually stopped.

1 5. Use of Bash scripts in a Linux OS.

- Document the use of bash scripts using a Linux OS.
- Give at least two examples with brief explanation and show screenshots.

Bash script in Linux, a powerful aspect of Linux systems, is a crucial tool in systems administration and programming (Grey, 2016). The script's commands are executed from top to bottom. Variables, conditionals, loops, and functions are also usable, significantly enhancing a script's functionality.

Bash scripts often make use of Linux's command-line tools. Combining these tools using pipes and redirection can create complex data processing pipelines (Shotts, 2019).

In my first example, I create a bash where the system asks me to introduce a number, and if that number is divided by 5, to divide the number and display the result

```
>_ main.sh x +
>_ main.sh
1  isDivisibleBy5() {
2      local number=$1
3
4      if (( number % 5 == 0 )); then
5          echo "$number is divisible by 5."
6          echo "Result of division: $((number / 5))"
7      else
8          echo "$number is not divisible by 5."
9      fi
10 }
11
12 echo "Enter a number:"
13 read number
14 isDivisibleBy5 $number
```

And the result:

```
>_ Console x Shell x +
>_ bash main.sh
Enter a number:
2775
2775 is divisible by 5.
Result of division: 555
>_
```

```
>_ Console x Shell x +
>_ bash main.sh
Enter a number:
287
287 is not divisible by 5.
>_
```

isDivisibleBy5() {: This line defines a function named `isDivisibleBy5`. The opening curly brace `{` starts the body of the function.

local number=\$1: This line declares a local variable `number` inside the function and assigns it the value of `$1`, which is the first argument passed to the function.

if ((number % 5 == 0)); then: This line starts an if-else control structure that checks if the number is divisible by 5.

echo "\$number is divisible by 5.": If the if condition is true, then this line is executed.

echo "Result of division: \$((number / 5))": This line prints the result of the division of the number by 5.

else: This keyword starts the else block of the if-else structure. If the if condition was false, the commands in this block are executed.

echo "\$number is not divisible by 5.": If the number is not divisible by 5, this line outputs a string stating that fact.

fi: This keyword ends the if-else structure.

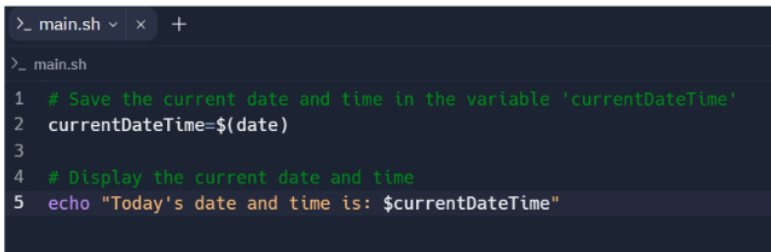
}; End of the function body.

echo "Enter a number:": This line prompts the user to enter a number.

read number: This line reads the input from the user and assigns it to the variable `number`.

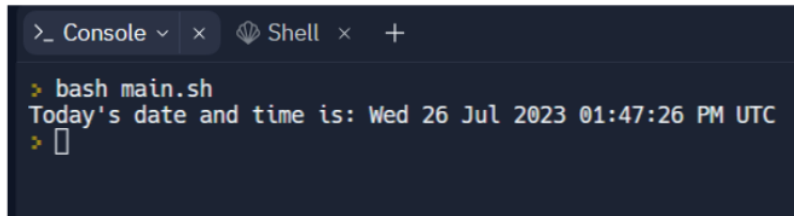
isDivisibleBy5 \$number: This line calls the function `isDivisibleBy5` with the user's input (`number`) as an argument. The script will then check if the input number is divisible by 5 and will print the corresponding message.

In my second example, the bash will display the current date and time:

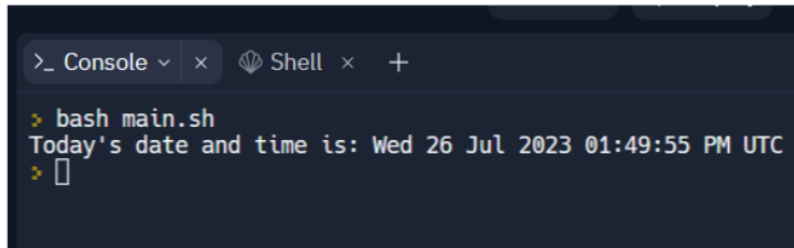


```
>_ main.sh x +
>_ main.sh
1 # Save the current date and time in the variable 'currentDateTime'
2 currentDateTime=$(date)
3
4 # Display the current date and time
5 echo "Today's date and time is: $currentDateTime"
```

And the result is:

A terminal window with a dark background. The title bar shows a dropdown menu with '>_ Console', a close button 'x', a shell icon, the text 'Shell', another close button 'x', and a plus sign '+'. The terminal content shows a prompt '>' followed by 'bash main.sh', then the output 'Today's date and time is: Wed 26 Jul 2023 01:47:26 PM UTC', and a final prompt '>' with a cursor.

And a new result, after two minutes:

A terminal window with a dark background. The title bar shows a dropdown menu with '>_ Console', a close button 'x', a shell icon, the text 'Shell', another close button 'x', and a plus sign '+'. The terminal content shows a prompt '>' followed by 'bash main.sh', then the output 'Today's date and time is: Wed 26 Jul 2023 01:49:55 PM UTC', and a final prompt '>' with a cursor.

currentDateTime=\$(date) uses the `date` command to get the current date and time and stores the result in the **currentDateTime** variable.

echo "Today's date and time is: \$currentDateTime" prints out a message that includes the current date and time stored in the **currentDateTime** variable.

Conclusion:

In conclusion, the journey through understanding operating systems and their functionalities has been quite enlightening. I've come to appreciate the vital role that an operating system plays in a computing system. It essentially acts as an intermediary between the hardware and the user. I've observed how they've evolved from basic single-tasking systems to today's multitasking, multi-user, real-time, and distributed systems, which clearly reflects the technological advancement and the escalating demands of users.

Through this exploration, I've learned how ¹⁴ different types of operating systems - single-user, multi-user, distributed, real-time, or multitasking - cater to distinct requirements and use cases. This flexibility allows us to choose the most appropriate OS based on our specific needs. Moreover, the ¹ functions of modern general operating systems like process management, memory management, file system management, and I/O system management have evidently become more refined and efficient over time.

Delving into the practical uses of operating systems, I found CLI commands to be a powerful tool for both Windows and Linux users. Despite the prevalence and user-friendliness of GUIs, CLI commands offer greater control and flexibility, enabling the execution of complex tasks more efficiently. The usage of CLI commands varies between Windows and Linux due to their distinct architectures and command syntax, but I've discovered that mastering these commands can greatly enhance my ability to navigate, manipulate, and use my operating system.

Similarly, I've learned that scripting in Windows and Linux provides automation capabilities that can lessen repetitive tasks and boost productivity. While Windows uses batch scripts, Linux employs bash scripts. The syntax and commands differ, but the principle remains the same: automating tasks to save time and effort.

In summary, this exploration of operating systems, their evolution, functionalities, and practical uses like CLI commands and scripting, has provided me with a comprehensive understanding of their significance in the world of computing. Even though the concepts may seem disparate at times, they all intertwine to provide a robust, efficient, and customizable computing environment. This has emphasized to me that continual learning and adaptation are essential to keep up with the rapid changes and improvements in the field.

Reference list:

1. Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2014). Operating Systems: Three Easy Pieces. Arpaci-Dusseau Books.
2. Barrett, D. J. (2020). Linux pocket guide: essential commands. O'Reilly Media, Inc.
3. Bovet, D. P., & Cesati, M. (2005). Understanding the Linux Kernel. O'Reilly Media.
4. Cooper, M. (2001). Advanced Bash-Scripting Guide. Linux Documentation Project.
5. Flylib. (n.d.). Batch Files and Batch Commands. [online] Available at: <https://flylib.com/books/en/4.477.1.280/1/> (Accessed: 22 July 2023).
6. Frish, R. (2016). Windows 10 For Dummies. John Wiley & Sons.
7. Grey, B. (2016). Learning the Bash Shell: Unix Shell Programming. O'Reilly Media.
8. McHoes, A. M., & Flynn, I. M. (2014). Understanding Operating Systems. Cengage Learning.
9. Microsoft. (2022). Rename. [online] Available at: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/rename> (Accessed: 22 July 2023).
10. Nemeth, E., Snyder, G., Hein, T. R., & Whaley, B. (2011). Unix and Linux system administration handbook. Pearson.
11. Shotts, W. (2019). The Linux command line: a complete introduction. No Starch Press.
12. Sosinsky, B. (2011). Networking Bible. John Wiley & Sons.
13. Stallings, W., & Brown, L. (2018). Computer Systems Architecture. Pearson.
14. Stanek, W. R. (2014). Windows Command-Line Administrator's Pocket Consultant. Microsoft Press.
15. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts. Wiley.
16. Tanenbaum, A. S. (2015). Modern Operating Systems. Pearson.

Assessment 2

ORIGINALITY REPORT

21 %
SIMILARITY INDEX

7 %
INTERNET SOURCES

2 %
PUBLICATIONS

19 %
STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Wales, Lampeter Student Paper	15 %
2	www.teachmint.com Internet Source	1 %
3	Submitted to Victoria University Student Paper	1 %
4	Submitted to University of Bolton Student Paper	1 %
5	link.springer.com Internet Source	<1 %
6	Submitted to Delta Career Education Corporation (Master) Student Paper	<1 %
7	www.coursehero.com Internet Source	<1 %
8	computerresearch.org Internet Source	<1 %
9	Submitted to Edith Cowan University Student Paper	<1 %

10	1lib.education Internet Source	<1 %
11	www.geeksforgeeks.org Internet Source	<1 %
12	www.scribd.com Internet Source	<1 %
13	www.slideshare.net Internet Source	<1 %
14	www.cisnigeria.com Internet Source	<1 %
15	cycadzblogs.home.blog Internet Source	<1 %
16	in.1lib.to Internet Source	<1 %

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off