

Assignment Submission - 2003836

by Jacob John Hopkins

FILE	24684_JACOB_JOHN_HOPKINS_ASSIGNMENT_SUBMISSION_- _2003836_331893_526229394.DOCX (1.45M)		
TIME SUBMITTED	25-MAR-2021 08:16PM (UTC+0000)	WORD COUNT	3820
SUBMISSION ID	147982588	CHARACTER COUNT	19900

Computer Architecture and Operating Systems

Assignment 1: Computing Portfolio

Jacob Hopkins

2003836

BSC Computing (Computer Networks and Cybersecurity)

Submission Deadline: Thursday March 25th

Table of Contents

Contents

Table of Contents 2

Introduction 3

Key Functions of General Operating Systems..... 3

 Memory Management 3

 Process Management 3

 System Security..... 4

 I/O Devices 4

 File Management 4

Use of CLI Commands in a Windows OS 4

Use of CLI Commands in a Linux OS..... 9

Use of Batch Scripts in a Windows OS 12

Use of Bash Scripts in a Linux OS 15

Conclusion..... 18

Reference List..... 19

Appendices..... 20

Introduction

Throughout this report, we will identify and analyse the key functions of general operating systems used widely today. Documenting how functions such as *memory* and *process* management occur, whilst also delving into system security and other raw interactions with an OS such as through the use of *command-line interface* commands. Each of these accounts will be supported with appropriately sourced material such as images and live examples for the benefit of the readers understanding.

Key Functions of General Operating Systems

An operating system, otherwise referred to as an **OS**, is the software used by a system to manage a device and its peripherals. This includes system resources, services and hardware.

Memory Management

When managing memory, the OS makes use of both the computer's RAM (main memory) and storage drive (secondary memory) to handle memory. Thanks to its fast serial presence detect (SPD) ^[10] speeds, RAM is the only storage medium for operating programs and processes. Copying data from secondary memory when accessed, the RAM withholds the necessary programs for the CPU and its processors to quickly access when in use. The memory handled by RAM is consistently monitored by the OS, altering the allocated memory for processes based on their usage and activity. When the RAM capacity is exceeded, the OS can temporarily create virtual memory (VRAM) within the secondary memory. This VRAM acts as additional RAM for the system, however due to the traditionally slower speeds of secondary memory in comparison to RAM, the programs stored can take longer to load.

Process Management

The OS handles the creation, schedule and termination of the programs and processes of the CPU. This is done through the use of *process control blocks* (PCB), which are controlled and managed by the CPU.



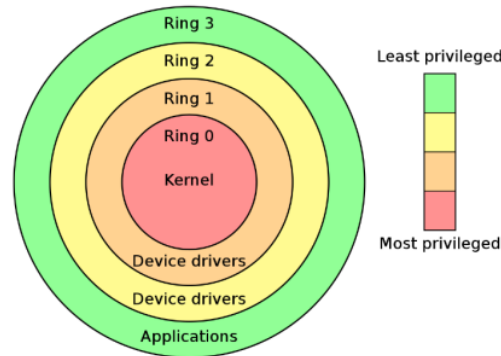
Process Control Block (PCB) [11]

To help identify each PCB, they are equipped with process IDs (PID). All necessary information on the process for the OS is stored within the PCB. To prevent damaging alterations to any PCBs, they are stored within secure memory such as the Kernel.

Please see **Appendix A** for more information on the memory layout of a process.

System Security

The OS has multiple ways of securing its host device, protecting it from potentially malicious users or programs. For direct access, users are required to enter credentials before accessing the device, this is done in various ways such as with a login and password. To help defend against malicious software, anti-virus and firewalls are used. To maintain OS stability, essential files are stored within secure memory out of reach of the user. This system is known as the *protection ring*, which is broken down into multiple rings each offering varying levels of access.



[13]

I/O Devices

Using specially built drivers, the OS can allow for the communication between I/O devices and its host computer. Depending on its usage, the OS can allocate and de-allocate a device from the CPU through its I/O traffic controller and scheduler for use by programs.

File Management

Depending on the operating system in question there are numerous types of file systems. Some examples of these file systems include Window's **FAT** (File Allocation Table), Linux's **ext** (Extended File System), and Apple's **APFS** (Apple File System).

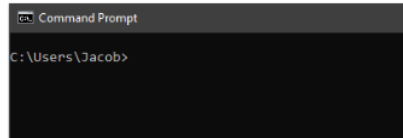
Servers have to “...serve multiple users at once over a network...”^[2] whilst they “...allow the users to share hardware and software resources”^[2]. This form of file management is known as *Distributed File Systems* (DFS), which as mentioned allows for the shared access of hardware and software, including files, and collaboration by different devices. Some examples of server OS that allow DFS include Windows Server and Red Hat Enterprise.

Please see **Appendix B** for more information on some of Windows' file systems, their advantages and disadvantages.

Use of CLI Commands in a Windows OS

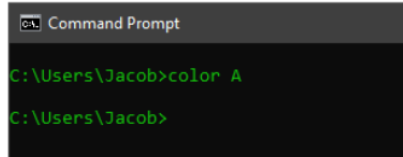
As described by Prof. Andrew Tanenbaum, “The shell command-line interface is much faster to use, more powerful, easily extensible, and does not give the user RSI from having to use a mouse all the time”^[3]. It is for these reasons many individuals remain loyal to the CLI rather than using a GUI.

The '**color**' command is a great tool to alter the appearance of the CLI and start off with. Typing '**color**' followed by either a number to change the background colour, or a letter to change the text colour. The default colours are a white font on a black background as seen below.



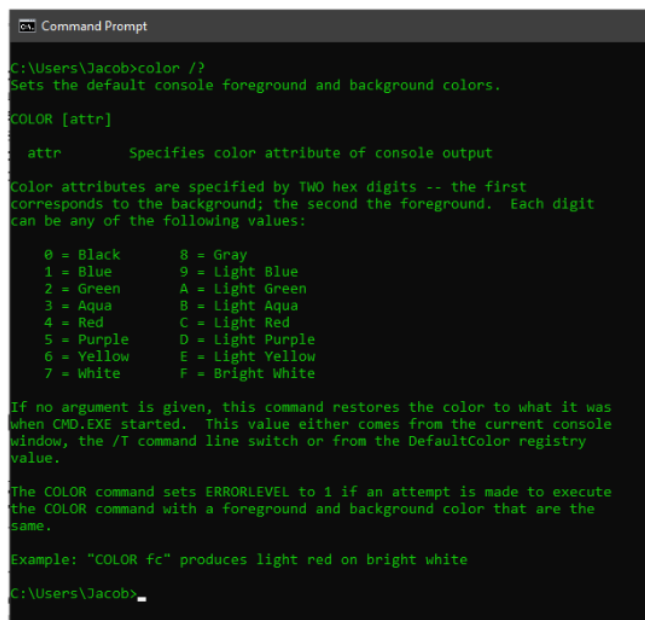
```
Command Prompt
C:\Users\Jacob>
```

Using the command '**color A**' will change our font colour to a retro CLI green.



```
Command Prompt
C:\Users\Jacob>color A
C:\Users\Jacob>
```

To learn what colours the numbers and letters represent, simply input the command '**color /?**' for help. A detailed list of the colour codes as well as details about the '**color**' command will be displayed.



```
Command Prompt
C:\Users\Jacob>color /?
Sets the default console foreground and background colors.

COLOR [attr]

    attr      Specifies color attribute of console output

Color attributes are specified by TWO hex digits -- the first
corresponds to the background; the second the foreground.  Each digit
can be any of the following values:

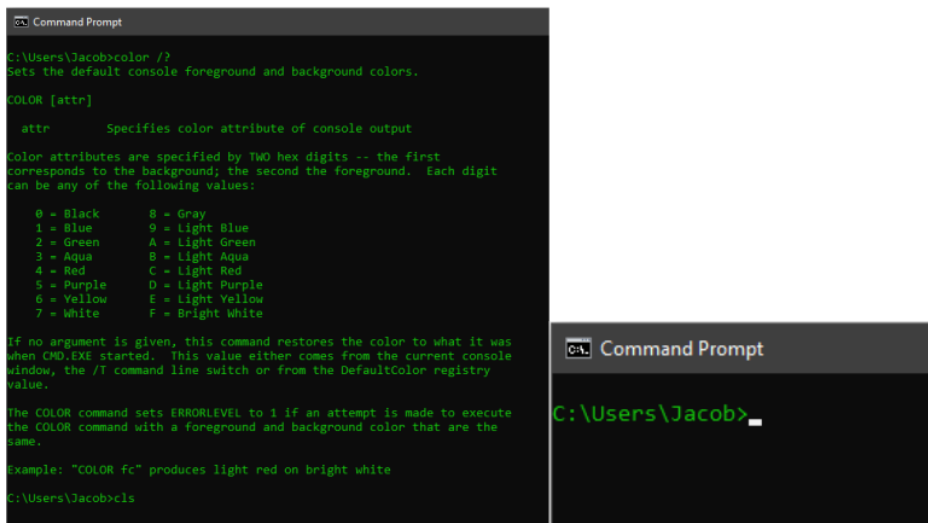
    0 = Black      8 = Gray
    1 = Blue       9 = Light Blue
    2 = Green      A = Light Green
    3 = Aqua       B = Light Aqua
    4 = Red        C = Light Red
    5 = Purple     D = Light Purple
    6 = Yellow     E = Light Yellow
    7 = White      F = Bright White

If no argument is given, this command restores the color to what it was
when CMD.EXE started.  This value either comes from the current console
window, the /T command line switch or from the DefaultColor registry
value.

The COLOR command sets ERRORLEVEL to 1 if an attempt is made to execute
the COLOR command with a foreground and background color that are the
same.

Example: "COLOR fc" produces light red on bright white
C:\Users\Jacob>
```

If we wanted to continue our use of the CLI within the same window but no longer needed the colour help list, we could use a useful command to clear the terminal. That command is '**cls**', which is short for clear screen.



```
C:\Users\Jacob>color /?
Sets the default console foreground and background colors.

COLOR [attr]

    attr                Specifies color attribute of console output

Color attributes are specified by two hex digits -- the first
corresponds to the background, the second the foreground. Each digit
can be any of the following values:

    0 = Black           8 = Gray
    1 = Blue            9 = Light Blue
    2 = Green           A = Light Green
    3 = Aqua            B = Light Aqua
    4 = Red             C = Light Red
    5 = Purple          D = Light Purple
    6 = Yellow          E = Light Yellow
    7 = White           F = Bright White

If no argument is given, this command restores the color to what it was
when CMD.EXE started. This value either comes from the current console
window, the /T command line switch or from the DefaultColor registry
value.

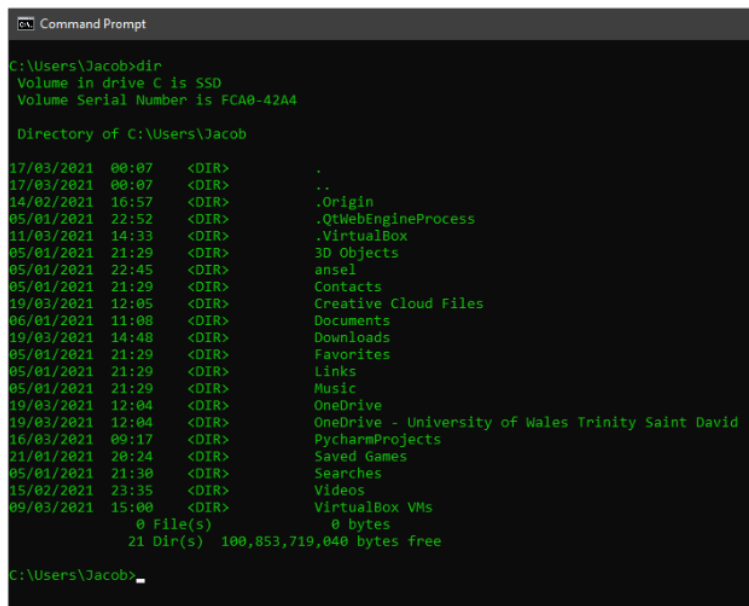
The COLOR command sets ERRORLEVEL to 1 if an attempt is made to execute
the COLOR command with a foreground and background color that are the
same.

Example: "COLOR fc" produces light red on bright white

C:\Users\Jacob>cls
```

Upon hitting enter, we can see the previous code has been cleared from the window.

Moving on from the **'color'** command, we can now begin to manoeuvre through our device and its directories. To check what directories we have available to use within our current directory we type the command **'dir'**, short for directory. This will display the directories within a list for us to view.



```
C:\Users\Jacob>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob

17/03/2021  00:07    <DIR>          .
17/03/2021  00:07    <DIR>          ..
14/02/2021  16:57    <DIR>          .Origin
05/01/2021  22:52    <DIR>          .QtWebEngineProcess
11/03/2021  14:33    <DIR>          .VirtualBox
05/01/2021  21:29    <DIR>          3D Objects
05/01/2021  22:45    <DIR>          ansel
05/01/2021  21:29    <DIR>          Contacts
19/03/2021  12:05    <DIR>          Creative Cloud Files
06/01/2021  11:08    <DIR>          Documents
19/03/2021  14:48    <DIR>          Downloads
05/01/2021  21:29    <DIR>          Favorites
05/01/2021  21:29    <DIR>          Links
05/01/2021  21:29    <DIR>          Music
19/03/2021  12:04    <DIR>          OneDrive
19/03/2021  12:04    <DIR>          OneDrive - University of Wales Trinity Saint David
16/03/2021  09:17    <DIR>          PycharmProjects
21/01/2021  20:24    <DIR>          Saved Games
05/01/2021  21:30    <DIR>          Searches
15/02/2021  23:35    <DIR>          Videos
09/03/2021  15:00    <DIR>          VirtualBox VMs
               0 File(s)              0 bytes
               21 Dir(s) 100,853,719,040 bytes free

C:\Users\Jacob>
```

If we see a directory we want to enter we need to use the **'cd'** command followed by the directory name, this will change the directory to your chosen one as implied by the abbreviated code. Here we will change to the downloads directory.

```
Command Prompt

C:\Users\Jacob>cd Downloads
C:\Users\Jacob\Downloads>_
```

Within downloads I would like to create a new folder for any images I download online. To do this we type the command '**mkdir**' followed by the name we would like to call the new folder/directory. In this instance we will call it images, to help make sure the directory has been created we can use the '**dir**' command to view all accessible directories within downloads.

```
Command Prompt

C:\Users\Jacob\Downloads>mkdir Images
C:\Users\Jacob\Downloads>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads

19/03/2021  16:29    <DIR>          .
19/03/2021  16:29    <DIR>          ..
19/03/2021  16:29    <DIR>          Images
               0 File(s)              0 bytes
               3 Dir(s)  100,618,117,120 bytes free

C:\Users\Jacob\Downloads>_
```

Now inside the images folder, I may want to store various image formats. However, if I wanted to only see images with the .png format I could use the command '**dir *png**'.

```
Command Prompt

C:\Users\Jacob\Downloads\Images>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads\Images

19/03/2021  16:38    <DIR>          .
19/03/2021  16:38    <DIR>          ..
19/03/2021  16:37             661,116 flower.png
19/03/2021  16:37             661,116 frog.jpg
19/03/2021  16:37             661,116 tree.png
               3 File(s)              1,983,348 bytes
               2 Dir(s)  100,621,316,096 bytes free

C:\Users\Jacob\Downloads\Images>dir *.png
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads\Images

19/03/2021  16:37             661,116 flower.png
19/03/2021  16:37             661,116 tree.png
               2 File(s)              1,322,232 bytes
               0 Dir(s)  100,622,282,752 bytes free

C:\Users\Jacob\Downloads\Images>_
```

If we want to delete something from a directory we can use the '**del**' command followed by the file name. Here we will delete the file frog.jpg.^[4]

```
Command Prompt
C:\Users\Jacob\Downloads\Images>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads\Images

19/03/2021  17:22    <DIR>          .
19/03/2021  17:22    <DIR>          ..
19/03/2021  16:37             661,116 flower.png
19/03/2021  16:37             661,116 frog.jpg
19/03/2021  16:37             661,116 tree.png
               3 File(s)              1,983,348 bytes
               2 Dir(s)  100,626,505,728 bytes free

C:\Users\Jacob\Downloads\Images>del frog.jpg

C:\Users\Jacob\Downloads\Images>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads\Images

19/03/2021  17:23    <DIR>          .
19/03/2021  17:23    <DIR>          ..
19/03/2021  16:37             661,116 flower.png
19/03/2021  16:37             661,116 tree.png
               2 File(s)              1,322,232 bytes
               2 Dir(s)  100,626,972,672 bytes free

C:\Users\Jacob\Downloads\Images>
```

If we wanted to leave a message for anyone looking for the deleted image to tell them it has been removed, we can use the '**echo**' command paired with '>' and the name we wish to give the file to write to a file.^[5]

```
Command Prompt
C:\Users\Jacob\Downloads\Images>echo Sorry I deleted the frog image > readme.txt

C:\Users\Jacob\Downloads\Images>dir
Volume in drive C is SSD
Volume Serial Number is FCA0-42A4

Directory of C:\Users\Jacob\Downloads\Images

19/03/2021  17:37    <DIR>          .
19/03/2021  17:37    <DIR>          ..
19/03/2021  16:37             661,116 flower.png
19/03/2021  17:37                33 readme.txt
19/03/2021  16:37             661,116 tree.png
               3 File(s)              1,322,265 bytes
               2 Dir(s)  100,626,853,888 bytes free

C:\Users\Jacob\Downloads\Images>
```

Once we are finished with Windows CLI, we can terminate the CLI terminal with the command '**exit**'.

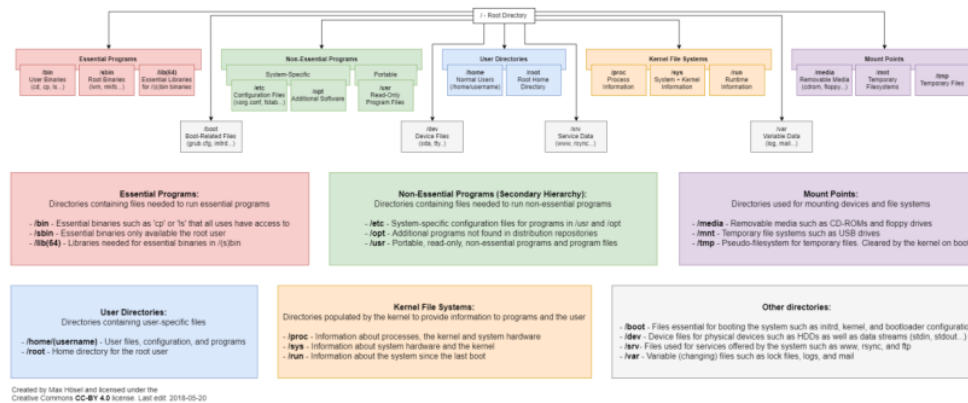
```
Command Prompt

C:\Users\Jacob>exit
```

Use of CLI Commands in a Linux OS

A fantastic illustration by Max Hösel depicts the Linux file system and accurately illustrates the system's tree structure at work. [6]

The Filesystem Hierarchy Standard (FHS)



When using any command-line interface, it is wise to establish which directory you are in before performing any other scripts. To do this, we can execute the command **'pwd'** which stands for **print working directory**. This command will print your current working directory for you to see.

```
root@kali:~# pwd
/root
root@kali:~#
```

Now knowing which directory we are working from; we are able to look at what other directories are closely linked for us to move into. For this we use the **'ls'** command, which will list our directory system.

```
root@kali: ~ 82x25
root@kali:~# ls
Desktop  Downloads  Pictures  sslstrip  Videos
Documents Music      Public    Templates
root@kali:~#
```

Just like Windows, to move through directories we use the **'cd'** command followed by the name of our desired directory. To then go back a directory we use **'cd ..'**. If we move into the Desktop directory with **'cd Desktop'** we can find a file called "readme.txt" using **'ls'**. The file contains some text, to have the CLI to display the text inside the file to us we use **'cat'** followed by the desired file name.

```
root@kali: ~/Desktop 82x25
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
readme.txt
root@kali:~/Desktop# cat readme.txt
Hello World!
root@kali:~/Desktop#
```

If we wanted to remove the “readme.txt” file from the Desktop, we would use a slightly different command to what is used in Windows CLI. In Linux we would use the ‘rm’ command with the name of the file we want to remove. Just like in Windows, we check the file is deleted by asking the terminal to list us the contents of the directory again using ‘ls’.

```
root@kali: ~/Desktop 82x25
root@kali:~/Desktop# ls
readme.txt
root@kali:~/Desktop# rm readme.txt
root@kali:~/Desktop# ls
root@kali:~/Desktop#
```

For more powerful user privileges, users must call for them using the ‘sudo’ command. This command however is only available to system admins and will require the admin password to execute.

```
root@kali: ~ 82x25
root@kali:~# sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
[command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-T timeout] [-u user] [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-T timeout] [-u user] file ...
```

To clear the terminal without closing the window, we use the ‘clear’ command.

```
root@kali: ~ 82x25
root@kali:~# sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
[command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-T timeout] [-u user] [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-T timeout] [-u user] file ...
root@kali:~# clear
```

After clearing the terminal, some users may want to view any previous commands executed for various reasons such as for security. This can be done using the ‘history’ command. When entered the terminal will return any of the previously executed code in a simple and easy to understand list format.

```

root@kali: ~ 82x25
root@kali:~# history
1  git clone https://github.com/noxx1/sslstrip.git && cd sslstrip && python se
tup.py install
2  cd # && rm -rf /usr/share/sslstrip/ && mv /root/sslstrip /usr/share/
3  sslstrip
4  man sslstrip
5  sudo apt autoremove
6  apt update && apt install firmware-ralink --no-upgrade -y
7  apt purge reaver -y autoremove && apt update && apt install sqlite3 libsqli
te3-dev libpcap0.8-dev --no-upgrade -y
8  apt purge reaver -y --autoremove && apt update && apt install sqlite3 libsq
lite3-dev libpcap0.8-dev --no-upgrade -y
9  git clone https://github.com/t6x/reaver-wps-fork-t6x && cd reaver-wps-fork-
t6x/src && ./configure && make && make install && cd ../.. && rm -rf reaver-wps-fo
rk-t6x/
10 clear
11 sslstrip
12 apt autoremove
13 pwd
14 chmod +x reaver
15 reaver
16 color A
17 cls
18 w
19 ls

```

For any additional information on commands within the CLI, you are able to call on a manual to print detailed information on any command within the CLI library. Typing '*man*' followed by the command will output your information. For this example I have written '*man sudo*'.

```

root@kali: ~ 82x25
SUDO(8) BSD System Manager's Manual SUDO(8)

NAME
    sudo, sudoedit - execute a command as another user

SYNOPSIS
    sudo -h | -K | -k | -V
    sudo -v [-ABknS] [-g group] [-h host] [-p prompt] [-u user]
    sudo -l [-ABknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
    [command]
    sudo [-ABEHnPS] [-C num] [-g group] [-h host] [-p prompt] [-r role]
    [-t type] [-T timeout] [-u user] [VAR=value] [-i | -s] [command]
    sudoedit [-ABknS] [-C num] [-g group] [-h host] [-p prompt] [-T timeout]
    [-u user] file ...

DESCRIPTION
    sudo allows a permitted user to execute a command as the superuser or an-
    other user, as specified by the security policy. The invoking user's real
    (not effective) user-ID is used to determine the user name with which to
    query the security policy.

    sudo supports a plugin architecture for security policies and input/output
    logging. Third parties can develop and distribute their own policy and
    I/O logging plugins to work seamlessly with the sudo front end. The de-
    Manual page sudo(8) line 1 (press h for help or q to quit)

```

Those using '*sudo*' may handle more advanced code that can include variables. Variables are simply a "memory *location*...that has a name that is associated with that location" [7]. We can create a variable by assigning a value to a name.

```

root@kali:~# myVariable=HelloWorld!
root@kali:~# echo $myVariable
HelloWorld!
root@kali:~# █

```

As seen in the code above, to print the value of a variable we must input a '\$' before our variable name. To then exit our CLI terminal, we simply enter '*exit*'.

```

root@kali:~# exit

```

Use of Batch Scripts in a Windows OS

Batch is a language used for creating programs to be executed within the Windows CLI. This particular scripting language uses the **'.bat'** extension. System admins often use batch scripts for various tasks and operations.

The following code demonstrates commands used within batch scripts, whilst also providing a brief explanation of the inner workings of the script.

```
BatScriptsExercise.bat
1  @Echo off
2  :: Changes title of cli window for user to know the code they're running
3  Title BatScriptsExercise
4
5  :: A collection of variables that are used and referenced throughout the script
6  set studentID=2003836
7  set list=Documents Downloads Videos
8
9  :: Script creates variable from user input
10 SET /P userName= Please enter your username:
11 IF "%userName%"=="Jacob" GOTO :authorisedUser
12
13 @Echo(
14 ECHO Unauthorised User
15 GOTO :end
16
17 :authorisedUser
18 @Echo(
19 ECHO Welcome Jacob!
20 SET /P userID= To confirm your identity, please enter your student ID:
21 :: If statement checks whether condition is true, if it is it will execute the desired script
22 IF "%userID%"=="%studentID%" GOTO :identityConfirmed
23
24 @Echo(
25 ECHO Incorrect ID, please try again
26 GOTO :authorisedUser
27
28 :identityConfirmed
29 ECHO Identity confirmed, creating backup of the following directories...
30 @Echo(
31 :: For loop prints contents of list for user
32 For %x In (%list%) Do (
33     Echo %x
34 )
35 @Echo(
36 Call :dirBackup
37 @Echo(
38 SET /P userChoice2= Would you like to run the backup again? (y/n)
39 IF "%userChoice2%"=="y" GOTO :identityConfirmed
40 IF "%userChoice2%"=="n" GOTO :end
41
42 ECHO Invalid choice
43 GOTO :end
44
45 :: Function is used to avoid rewritten code
46 :dirBackup
47 ECHO Backup of %list% started...
48 :: xcopy is built into windows and is used to create a copy of files/folders and paste them elsewhere
49 xcopy /s C:\Users\Jacob\Documents E:\Backup
50 xcopy /s C:\Users\Jacob\Downloads E:\Backup
51 xcopy /s C:\Users\Jacob\Videos E:\Backup
52
53 EXIT /B 0
54
55 @Echo(
56
57
58 :end
59 @Echo(
60 ECHO Thank you for using my batch backup program!
61 :: Script closes
62 Pause
```

Throughout the code, various comments have been placed to provide the reader/user with an understanding of the script and its process. Additionally, these comments can help with guiding the programmer through their own decisions if they need reminding in the future.

```

BatScriptsExercise
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Jacob>BatScriptsExercise.bat
Please enter your username:Jacob

Welcome Jacob!
To confirm your identity, please enter your student ID:2003836
Identity confirmed, creating backup of the following directories...

Documents
Downloads
Videos

Backup of Documents Downloads Videos started...
Does E:\Backup specify a file name
or directory name on the target
(F = file, D = directory)? _

```

The script begins by declaring the title for the script for the user to see. It then asks for input from the user to store as a variable and compare against existing data.

```

2  :: Changes title of cli window for user to know the code they're running
3  Title BatScriptsExercise

9  :: Script creates variable from user input
10 SET /P userName= Please enter your username:
11 :: IF statement checks whether condition is true, if it is it will execute the desired script
12 :: otherwise the script will terminate
13 IF "%userName%"=="Jacob" GOTO :authorisedUser
14
15 @Echo(
16 ECHO Unauthorised User
17 GOTO :end

58 :end
59 @Echo(
60 ECHO Thank you for using my batch backup program!
61 :: Script closes
62 Pause

```

A *For loop* is also used within the script to print a list variable for the user, eliminating the need for excessive amounts of code.

```

7  set list=Documents Downloads Videos

31 :: For loop prints contents of list for user
32 For %%x In (%list%) Do (
33     Echo %%x)

```

The script finalises by creating a copy of the directories within the list variable and stores the copies within a separate drive, creating a backup of those files. The backup is done using the '*xcopy*' command. ^[8]

```

45 :dirBackup
46 ECHO Backup of %list% started...
47 :: xcopy is built into windows and is used to create a copy of files/folders and paste them elsewhere
48 xcopy /s C:\Users\Jacob\Documents E:\Backup
49 xcopy /s C:\Users\Jacob\Downloads E:\Backup
50 xcopy /s C:\Users\Jacob\Videos E:\Backup
51
52 EXIT /B 0

```

Before exiting however, the script asks the user one last time whether they would like to run the program again, backing up their files. If the user replies 'yes' the program code will loop back to where the backup commences. Otherwise, the script will terminate.

```

39 SET /P userChoice2= Would you like to run the backup again? (y/n)
40 IF "%userChoice2%"=="y" GOTO :identityConfirmed
41 IF "%userChoice2%"=="n" GOTO :end

```

```

C:\Users\Jacob>BatScriptsExercise.bat
Please enter your username:Jacob

Welcome Jacob!
To confirm your identity, please enter your student ID:2003836
Identity confirmed, creating backup of the following directories...

Documents
Downloads
Videos

Backup of Documents Downloads Videos started...
Does E:\Backup specify a file name
or directory name on the target
(F = file, D = directory)? d
0 File(s) copied
Does E:\Backup specify a file name
or directory name on the target
(F = file, D = directory)? d
0 File(s) copied
Does E:\Backup specify a file name
or directory name on the target
(F = file, D = directory)? d
0 File(s) copied

Would you like to run the backup again? (y/n)n

Thank you for using my batch backup program!
Press any key to continue . . .

```

For additional security, the script makes use of the variables thought the code to verify the user is authorised to access the systems data. Requests for usernames and IDs are made for authentication. If the entered data is incorrect, the user is simply turned away.

```

C:\Users\Jacob>BatScriptsExercise.bat
Please enter your username:Richard

Unauthorised User

Thank you for using my batch backup program!
Press any key to continue . . .

```

```

9  :: Script creates variable from user input
10 SET /P userName= Please enter your username:
11 IF "%userName%"=="Jacob" GOTO :authorisedUser
12
13 @Echo(
14 ECHO Unauthorised User
15 GOTO :end
16
17 :authorisedUser
18 @Echo(
19 ECHO Welcome Jacob!
20 SET /P userID= To confirm your identity, please enter your student ID:
21 :: IF statement checks whether condition is true, if it is it will execute the desired script
22 IF "%userID%"=="%studentID%" GOTO :identityConfirmed
23
24 @Echo(
25 ECHO Incorrect ID, please try again
26 GOTO :authorisedUser

```

Use of Bash Scripts in a Linux OS

Built to replace the existing Unix Bourne shell, the Bash language, short for Bourne Again Shell, was the work of Brian Fox released in 1989.

The following code demonstrates commands used within batch scripts, whilst also providing a brief explanation of the inner workings of the script.

```
1 #!/bin/bash
2
3 # function is self-contained code that can be called to for use throughout the script
4 userExperience () {
5     echo "Do you have any experience with Bash scripting?"
6     read -p "Yes/No?: " userAnswer
7     case $userAnswer in
8         Yes|yes|Y|y)
9         echo "Excellent"
10        ;;
11        No|no|N|n)
12        echo "That's unfortunate..."
13        ;;
14        *)
15        echo "Try again: enter Yes or No"
16        ;;
17    esac
18 }
19
20 echo "Create New User..."
21 read -p "Username: " userName
22 read -sp "Password: " userPassword
23
24 echo
25 echo "Welcome $userName, your password has been hidden to maintain confidentiality"
26 echo "Storing login credentials in backup file..."
27
28 # writing user entered credentials to text file for records
29 echo $userName > LoginDetails.txt
30 echo $userPassword >> LoginDetails.txt
31 echo "... done!"
32
33 echo
34 read -p "Please enter your age: " userAge
35
36 # if the user's age is -gt greater than 18 they're eligible, else they are not
37 if [ $userAge -gt 18 ];
38 then
39     echo "You are eligible to create this account!"
40     echo
41     userExperience
42 else
43     echo "Unfortunately you are not eligible to create this account!"
44     echo
45     exit
46 fi
47
48 # until loop will iteratively execute as long as the expression evaluates to false
49 until [ "$userGuess" = "Brian Fox" ]
50 do
51     echo "Let's test your knowledge with a quick question! Who developed bash?: "
52     read userGuess
53     if [ "$userGuess" != "Brian Fox" ]; then
54         echo -e "\e[1;31m << INCORRECT. Try Again. >> \e[0m"
55     fi
56 done
57
58 echo
59 echo "That's correct, well done!"
60
61 # arrays are like variables, only they can hold numerous values at once
62 # each of these variables can be called to at different times
63 declare -A Array1
64
65 Array1[Var1]="June"
66 Array1[Var2]="8th"
67 Array1[Var3]="1989"
68
69 echo "Bash was developed by Brian Fox on ${Array1[@]}"
70
71 echo
72 echo -e "\e[1;31m End of program. \e[0m"
```

Throughout the code, various commands such as *functions*, *variables*, *until loops*, and more have been utilised to produce the functioning program.

```
root@kali: ~ 98x65
root@kali:~# sh -e BashScriptsExercise.txt
Create New User...
Username: Jacob
Password:
Welcome Jacob, your password has been hidden to maintain confidentiality
Storing login credentials in backup file...
...done!

Please enter you age:
```

The script begins by prompting the user to create an account, this is done by storing the entered data from the user within variables for later use. As mentioned within the script, when entering the password, the script hides the entered data, this is done using the `'-sp'` command to withhold confidentiality. To further protect the account, the script also creates an external text file and prints the credentials of the user to it in case of a future emergency.

```
27 echo "Welcome $userName, your password has been hidden to maintain confidentiality"
28 echo "Storing login credentials in backup file..."
29
30 # writing user entered credentials to text file for records
31 echo $userName > LoginDetails.txt
32 echo $userPassword >> LoginDetails.txt
33 echo "... done!"
```

To ensure the user is eligible to create their account, an *If statement* has been created to test the age of the user. If the user is under the age of 18, they are not eligible to create their account, thus kicking them out of the program.

```
38 # if the user's age is -gt greater than 18 they're eligible, else they are not
39 if [ $userAge -gt 18 ];
40 then
41     echo "You are eligible to create this account!"
42     echo
43     userExperience
44 else
45     echo "Unfortunately you are not eligible to create this account!"
46     echo
47     exit
```

```
root@kali: ~ 73x52
root@kali:~# sh -e BashScriptsExercise.txt
Create New User...
Username: Jacob
Password:
Welcome Jacob, your password has been hidden to maintain confidentiality
Storing login credentials in backup file...
...done!

Please enter you age: 16
Unfortunately you are not eligible to create this account!

End of program.
root@kali:~#
```

The until loop will iteratively execute as long as the expression evaluates to false. In this instance, **until** the user inputs the correct answer to the scripts questions.

```

50 # until loop will iteratively execute as long as the expression evaluates to false
51 until [ "$userGuess" = "Brian Fox" ]
52 do
53     echo "Let's test your knowledge with a quick question! Who developed bash?: "
54     read userGuess
55     if [ "$userGuess" ≠ "Brian Fox" ]; then
56         echo -e "\e[1;31m << INCORRECT. Try Again. >> \e[0m"
57     fi
58 done

```

In this particular script, the function is housing a case statement. These case statements hold multiple variables which are compared against to deliver an outcome, here that outcome is whether the answer given by the user matches the correct answer stored in the code. If the user's guess does not match the variable held by the script, the user will be notified and prompted to try again until their guess matches.

```

root@kali: ~ 73x52
root@kali:~# sh -e BashScriptsExercise.txt
Create New User...
Username: Jacob
Password:
Welcome Jacob, your password has been hidden to maintain confidentiality
Storing login credentials in backup file...
...done!

Please enter your age: 20
You are eligible to create this account!

Do you have any experience with Bash scripting?
Yes/No?: Y

Excellent
Let's test your knowledge with a quick question! Who developed bash?:
Brian Cox
<< INCORRECT. Try Again. >>
Let's test your knowledge with a quick question! Who developed bash?:
Brian Fox

That's correct, well done!
Bash was developed by Brian Fox on June 8th 1989

End of program.
root@kali:~#

```

```

3 # function is self-contained code that can be called to for use throughout the script
4 userExperience () {
5     echo "Do you have any experience with Bash scripting?"
6     read -p "Yes/No?: " userAnswer
7     case $userAnswer in
8         Yes|yes|Y|y)
9         echo
10            echo "Excellent"
11            ;;
12         No|no|N|n)
13            echo
14            echo "That's unfortunate..."
15            ;;
16         *)
17            echo
18            echo "Try again: enter Yes or No"
19            ;;
20     esac
21 }

```

An array can hold multiple variables/values, with each of these variables/values able to be called to at any time. This particular array holds three variables, each with their own unique value. Within the script, the array is declared with a '-A' because of its use of words in addition to numbers, making it

an associative array. It is then called to output all of its variables; this is done using the `'${Array1[@]}'` command.

```
Bash was developed by Brian Fox on June 8th 1989

End of program.
root@kali:~# █

64 # arrays are like variables, only they can hold numerous values at once
65 # each of these variables can be called to at different times
66 declare -A Array1
67
68 Array1[Var1]="June"
69 Array1[Var2]="8th"
70 Array1[Var3]="1989"
71
72 echo "Bash was developed by Brian Fox on ${Array1[@]}"
```

Conclusion

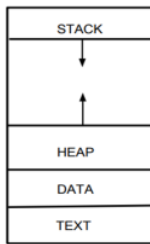
The operating system is a vastly powerful and sophisticated work of art. Managing many different aspects of its host, the OS takes control of most elements and handles them in ways that ensure the system can run at its best, whilst also minimising problems. It takes patience and determination to tame an OS using CLI scripting, but with the right skills you can create great pieces of code that can help make your CLI experience just that little bit easier.

Reference List

- [1] Microsoft, "Overview of FAT, HPFS, and NTFS File Systems", *Microsoft*, 2020. [Online] Available: <https://docs.microsoft.com/en-us/troubleshoot/windows-client/backup-and-storage/fat-hpfs-and-ntfs-file-systems>, [Accessed Mar. 18, 2021].
- [2] A. Tanenbaum, and H. Bos, *Modern Operating Systems*, 4th ed., Amsterdam: Pearson PLC, 2014, ch.1, pp.35-36.
- [3] A. Tanenbaum, and H. Bos, *Modern Operating Systems*, 4th ed., Amsterdam: Pearson PLC, 2014, ch.10, pp.726.
- [4] thenewboston, "Windows Command Line Tutorials", *YouTube*, Jan. 2016. [Online] Available: <https://www.youtube.com/playlist?list=PL6gx4Cwl9DGDV6SnbINIVUd0o2xT4JbMu> [Accessed Mar. 19, 2021].
- [5] joshtronic, "Command-line Basics: Creating Files and Directories", *alligator.io*. [Online] Available: <https://alligator.io/workflow/command-line-creating-files-directories/> [Accessed Mar. 19, 2021].
- [6] M. Hösel, "The Filesystem Hierarchy Standard (FHS)", *Reddit*, May. 20, 2018. [Online]. Available: https://www.reddit.com/r/linux/comments/8kt99k/the_file_system_hierarchy_standard_visualized_or/ [Accessed Mar. 22, 2021].
- [7] BBC, "Constants, variables and data types", *BBC Bitesize*. [Online] Available: <https://www.bbc.co.uk/bitesize/guides/zc6s4wx/revision/5#:~:text=Variables%20are%20data%20values%20that,variable%20is%20a%20memory%20location%20.&text=When%20data%20is%20read%20from,copied%20and%20used%20in%20calculations> [Accessed Mar. 23, 2021].
- [8] lavinio, "Batch file to copy files from on folder to another folder", *Stack Overflow*, Jun. 12, 2009. [Online]. Available: <https://stackoverflow.com/questions/986447/batch-file-to-copy-files-from-one-folder-to-another-folder> [Accessed Mar. 24, 2021].
- [9] RIP Tutorial, "Changing Directories and Listing their Contents", *RIP Tutorial*. [Online]. Available: <https://riptutorial.com/batch-file/topic/7132/changing-directories-and-listing-their-contents> [Accessed Mar. 24, 2021].
- [10] HAL_404, "For Those Who Don't Know What SPD memory speed is for", *[H]ardForum*, Jan. 4, 2020. [Online] Available: <https://hardforum.com/threads/for-those-who-dont-know-what-spd-memory-speed-is-for.1991293/#:~:text=%22When%20a%20computer%20is%20booted,%2C%20data%20width%2C%20speed%2C%20and> [Accessed Mar. 25, 2021]
- [11] A. Onsmán, "What is Process Control Block (PCB)?", *Tutorialspoint*, Sep. 20, 2018. [Online] Available: <https://www.tutorialspoint.com/what-is-process-control-block-pcb> [Accessed Mar. 25, 2021].
- [12] H. Singh Bisht, "Memory layout of a process | Operating System", *includehelp*, Sep. 27, 2018. [Online] Available: <https://www.includehelp.com/operating-systems/memory-layout-of-a-process.aspx> [Accessed Mar. 25, 2021].
- [13] Wikipedia, "Protection ring", *Wikipedia*. [Online] Available: https://en.wikipedia.org/wiki/Protection_ring [Accessed Mar. 25, 2021].

Appendices

Appendix A ^[12]



Stack Section

This section contains local variable, function and returns address. As in the diagram show stack and heap grow in opposite direction which is obvious if both grow in the same direction then they may overlap so it is good if they grow in opposite direction.

Heap Section

This Section is used to provide dynamic memory whenever memory is required by the program during runtime It is provided form heap section.

Data Section

This Section contains the global variables and static local variables.

Text Section

This section contains the executable instruction, constants, and macros, it is read-only location and is sharable so that can be used by another process also.

Appendix B ^[1]

FAT overview

FAT is by far the most simplistic of the file systems supported by Windows NT. The FAT file system is characterized by the file allocation table (FAT), which is really a table that resides at the very "top" of the volume. To protect the volume, two copies of the FAT are kept in case one becomes damaged. In addition, the FAT tables and the root directory must be stored in a fixed location so that the system's boot files can be correctly located.

A disk formatted with FAT is allocated in clusters, whose size is determined by the size of the volume. When a file is created, an entry is created in the directory and the first cluster number containing data is established. This entry in the FAT table either indicates that this is the last cluster of the file, or points to the next cluster.

Updating the FAT table is very important as well as time consuming. If the FAT table is not regularly updated, it can lead to data loss. It is time consuming because the disk read heads must be repositioned to the drive's logical track zero each time the FAT table is updated.

There is no organization to the FAT directory structure, and files are given the first open location on the drive. In addition, FAT supports only read-only, hidden, system, and archive file attributes.

Advantages of FAT

It is not possible to perform an undelete under Windows NT on any of the supported file systems. Undelete utilities try to directly access the hardware, which cannot be done under Windows NT. However, if the file was located on a FAT partition, and the system is restarted under MS-DOS, the file can be undeleted. The FAT file system is best for drives and/or partitions under approximately 200 MB, because FAT starts out with very little overhead. For further discussion of FAT advantages, see the following:

- Windows NT Server "Concepts and Planning Guide," Chapter 5, section titled "Choosing a File System".
- Windows NT Workstation 4.0 Resource Kit, Chapter 18, "Choosing a File System"
- Windows NT Server 4.0 Resource Kit "Resource Guide," Chapter 3, section titled "Which File System to Use on Which Volumes".

Disadvantages of FAT

Preferably, when using drives or partitions of over 200 MB the FAT file system should not be used. This is because as the size of the volume increases, performance with FAT will quickly decrease. It is not possible to set permissions on files that are FAT partitions.

FAT partitions are limited in size to a maximum of 4 Gigabytes (GB) under Windows NT and 2 GB in MS-DOS.

For further discussion of other disadvantages of FAT, see the following:

- Windows NT Server "Concepts and Planning Guide," Chapter 5, section titled "Choosing a File System".
- Windows NT Workstation 4.0 Resource Kit, Chapter 18, "Choosing a File System"
- Microsoft Windows NT Server 4.0 Resource Kit "Resource Guide," Chapter 3, section titled "Which File System to Use on Which Volumes".

NTFS overview

From a user's point of view, NTFS continues to organize files into directories, which, like HPFS, are sorted. However, unlike FAT or HPFS, there are no "special" objects on the disk and there is no dependence on the underlying hardware, such as 512-byte sectors. In addition, there are no special locations on the disk, such as FAT tables or HPFS Super Blocks.

The goals of NTFS are to provide:

- Reliability, which is especially desirable for high end systems and file servers.
- A platform for added functionality.
- Support POSIX requirements
- Removal of the limitations of the FAT and HPFS file systems

Remove Limitations

First, NTFS has greatly increased the size of files and volumes, so that they can now be up to 2^{64} bytes (16 exabytes or 18,446,744,073,709,551,616 bytes). NTFS has also returned to the FAT concept of clusters in order to avoid HPFS problem of a fixed sector size. This was done because Windows NT is a portable operating system and different disk technology is likely to be encountered at some point. Therefore, 512 bytes per sector was viewed as having a large possibility of not always being a good fit for the allocation. This was accomplished by allowing the cluster to be defined as multiples of the hardware's natural allocation size. Finally, in NTFS all filenames are Unicode based, and 8.3 filenames are kept along with long filenames.

Advantages of NTFS

NTFS is best for use on volumes of about 400 MB or more. This is because performance does not degrade under NTFS, as it does under FAT, with larger volume sizes.

The recoverability designed into NTFS is such that a user should never have to run any sort of disk repair utility on an NTFS partition. For additional advantages of NTFS, see the following:

- Windows NT Server "Concepts and Planning Guide," Chapter 5, section titled "Choosing a File System".
- Windows NT Workstation 4.0 Resource Kit, Chapter 18, "Choosing a File System"
- Windows NT Server 4.0 Resource Kit "Resource Guide," Chapter 3, section titled "Which File System to Use on Which Volumes".

Disadvantages of NTFS

It is not recommended to use NTFS on a volume that is smaller than approximately 400 MB, because of the amount of space overhead involved in NTFS. This space overhead is in the form of NTFS system files that typically use at least 4 MB of drive space on a 100-MB partition.

Currently, there is no file encryption built into NTFS. Therefore, someone can boot under MS-DOS, or another operating system, and use a low-level disk editing utility to view data stored on an NTFS volume.

It is not possible to format a floppy disk with the NTFS file system; Windows NT formats all floppy disks with the FAT file system because the overhead involved in NTFS will not fit onto a floppy disk.

For further discussion of NTFS disadvantages, see the following:

- Windows NT Server "Concepts and Planning Guide," Chapter 5, section titled "Choosing a File System".
- Windows NT Workstation 4.0 Resource Kit, Chapter 18, "Choosing a File System"
- Windows NT Server 4.0 Resource Kit "Resource Guide," Chapter 3, section titled "Which File System to Use on Which Volumes".

Assignment Submission - 2003836

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

92/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

GRADING FORM: UWTSD GRADING FORM

JACOB JOHN HOPKINS

92

1. RESEARCH LITERATURE

Extent of research and/or own reading, selection of credible sources, application of appropriate referencing conventions.



+ Very good use of citations. + formatted reference list. - 1/2 books and/or academic papers could be included in the reference list for additional marks.
Tip: Use multiple citations in places to sustain and enhance the grade for this category.

18

2. KNOWLEDGE AND UNDERSTANDING

Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.



Key functions: + Very good overview of key OS functions documented. Bat Commands: + Very good use of relevant commands with accompanying narration that demonstrates a sound understanding of the commands used.

19

Bash Commands: + Very good use of relevant commands with accompanying narration that demonstrates a sound understanding of the commands. Bat Scripts: + Very good use and understanding of relevant scripts documented. - could be extended to include the use of arrays. Note: demonstrated in Bash scripts. Good. Bash Scripts: + Very good use and understanding of relevant scripts documented.

3. ANALYSIS

Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence.



Evidence of some good analysis and understanding of OS functions and the relevant languages is clearly documented. + Conclusion.

9

4. PRACTICAL APPLICATION

Deployment of methods, materials, tools and techniques; application of concepts; formulation of innovative and creative solutions to solve problems.



1. Bat Commands: + Excellent use of relevant commands. 2. Bash Commands: + Excellent use of relevant commands. 3. Bat Scripts: + Very good use of relevant scripts to demonstrate knowledge of the core syntax. 4. Bash Scripts: + Very good use of relevant scripts to demonstrate knowledge of the core syntax.

38

5. PROFESSIONAL PRACTICE

*Attributes in professional practice: individual and collaborative working;
deployment of appropriate media; presentation and organisation.*



- + Good report format, structure, and presentation.
 - Report sections could be numbered. +/- Appendix:
- Good use of Appendices with relevant content.
Activity 1 answer could be included.

8