

# Assignment 1 - Operating Systems

Ben Roberts (2205382)

Computer Architecture and Operating Systems (ACCA4022)

University of Wales Trinity Saint David – School of Applied Computing

## Table of Contents

|                                                           |    |
|-----------------------------------------------------------|----|
| Introduction.....                                         | 5  |
| Task 1 – Key functions of general Operating Systems.....  | 6  |
| Task 2 – Use of Windows OS CLI commands.....              | 8  |
| Task 3 – Use of Linux OS CLI commands.....                | 12 |
| Task 4 – Use of Windows OS scripts.....                   | 15 |
| Task 5 – Use of Linux OS scripts.....                     | 22 |
| Conclusion.....                                           | 30 |
| Bibliography.....                                         | 31 |
| Appendices.....                                           | 34 |
| Appendix A – OS Tutorial 1.....                           | 34 |
| Appendix B – Certificate for Linux Essentials course..... | 35 |

## Table of Figures

|                                                                                                                                   |    |
|-----------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 1: htop utility showing active processes and system resource usage.....                                                    | 7  |
| Figure 2: Output of the Windows cd command.....                                                                                   | 8  |
| Figure 3: Printing "Hello World" with the echo command.....                                                                       | 8  |
| Figure 4: Output of the Windows dir command in a user's folder.....                                                               | 8  |
| Figure 5: Creating a directory with mkdir, moving in and out of it with cd, then deleting it with rmdir.....                      | 9  |
| Figure 6: List of commands produced by the Windows "help" command.....                                                            | 9  |
| Figure 7: Output of the Windows help command with echo as an argument.....                                                        | 9  |
| Figure 8: Performing file management operations on a Thumbs.db file with the Windows copy, del and move commands.....             | 10 |
| Figure 9: Outputting dummy text from a plain text file using the Windows type command.....                                        | 11 |
| Figure 10: Changing directories on Linux with the cd command.....                                                                 | 12 |
| Figure 11: Listing the contents of a game folder using the Bash ls command.....                                                   | 12 |
| Figure 12: Outputting "Hello World" with the Bash echo command.....                                                               | 12 |
| Figure 13: Performing file management on Linux with the Bash mv, cp and rm commands.....                                          | 12 |
| Figure 14: Creating and deleting a directory with the Bash mkdir and rmdir commands....                                           | 13 |
| Figure 15: Printing system information with the Bash uname command.....                                                           | 13 |
| Figure 16: Checking the manual page for the "man" command using the Bash man command.....                                         | 13 |
| Figure 17: Outputting the contents of a plain text file containing placeholder text with the Bash cat command.....                | 14 |
| Figure 18: Making a file executable with the Bash chmod command.....                                                              | 14 |
| Figure 19: Sample "Hello World" batch script.....                                                                                 | 15 |
| Figure 20: Output of the above hello world script.....                                                                            | 15 |
| Figure 21: Sample batch script utilising variables.....                                                                           | 15 |
| Figure 22: Output of the above variable script.....                                                                               | 15 |
| Figure 23: Sample batch file containing arithmetic operations.....                                                                | 16 |
| Figure 24: Output of the above arithmetic script.....                                                                             | 16 |
| Figure 25: Sample batch script with goto operations.....                                                                          | 17 |
| Figure 26: Output of the above script with goto operations.....                                                                   | 17 |
| Figure 27: Sample batch script with function operations.....                                                                      | 17 |
| Figure 28: Output of the above function script.....                                                                               | 17 |
| Figure 29: Sample batch script demonstrating if statements.....                                                                   | 18 |
| Figure 30: Output of the above script with statements.....                                                                        | 18 |
| Figure 31: Sample batch script containing a set command to store user input as a variable.....                                    | 19 |
| Figure 32: Output of the above script, demonstrating user input.....                                                              | 19 |
| Figure 33: Sample batch script using if and goto in order to simulate a dialogue confirmation.....                                | 19 |
| Figure 34: Output of the above script with user input, returning to an earlier point in the script when the option is denied..... | 19 |
| Figure 35: Sample batch script with a for loop.....                                                                               | 20 |
| Figure 36: Output of the above for loop.....                                                                                      | 20 |
| Figure 37: Improved version of the above script that auto-increments instead of iterating through a list.....                     | 20 |
| Figure 38: Output of the above array script.....                                                                                  | 21 |
| Figure 39: Sample batch script showing reading and writing from and to an array.....                                              | 21 |

|                                                                                                                                                     |    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 40: POSIX shebang pointing a script to the Bash shell interpreter.....                                                                       | 22 |
| Figure 41: Single-line Bash shell script comment prefixed with a hash symbol.....                                                                   | 22 |
| Figure 42: Bash shell script multiline comment.....                                                                                                 | 22 |
| Figure 43: Alternate method of adding a multiline comment in a Bash shell script.....                                                               | 23 |
| Figure 44: Sample Bash shell script demonstrating the use of variables with an echo command.....                                                    | 23 |
| Figure 45: Output of the above variable script.....                                                                                                 | 23 |
| Figure 46: Sample shell script to store and output user input.....                                                                                  | 23 |
| Figure 47: Output of the above input script.....                                                                                                    | 23 |
| Figure 48: Sample bash script for performing arithmetic.....                                                                                        | 24 |
| Figure 49: Output of the above arithmetic script.....                                                                                               | 24 |
| Figure 50: Shell script using an if statement to run different commands based on a variable.....                                                    | 24 |
| Figure 51: Output of the above script with if statements.....                                                                                       | 24 |
| Figure 52: Bash script showing the use of less than and greater than operations as well as elif to cycle through responses based on a variable..... | 25 |
| Figure 53: Output of the above shell script showing the result with a value of "8".....                                                             | 25 |
| Figure 54: Batch Script utilising case switching.....                                                                                               | 25 |
| Figure 55: Output of the above switch utilising case switching.....                                                                                 | 25 |
| Figure 56: Simple Bash shell script making use of a for loop.....                                                                                   | 26 |
| Figure 57: Output of the above for loop script.....                                                                                                 | 26 |
| Figure 58: Example Bash shell script with a while loop.....                                                                                         | 26 |
| Figure 59: Output of the above while loop script.....                                                                                               | 26 |
| Figure 60: Bash shell script containing an until loop.....                                                                                          | 27 |
| Figure 61: Output of the above until loop script.....                                                                                               | 27 |
| Figure 62: Bash echo command in a script being ran with the -e flag, enabling the use of escape characters for text formatting.....                 | 27 |
| Figure 63: Output of the above command.....                                                                                                         | 27 |
| Figure 64: A series of Bash echo commands with ANSO escape codes.....                                                                               | 27 |
| Figure 65: Output of the above formatting script.....                                                                                               | 27 |
| Figure 66: Bash script file utilising functions to enable text formatting with ANSI codes....                                                       | 28 |
| Figure 67: Output of the above script with ANSI formatting.....                                                                                     | 28 |
| Figure 68: Bash script displaying a variety of implementations for Arrays in Bash.....                                                              | 29 |
| Figure 69: Output of the above array script.....                                                                                                    | 29 |
| Figure 70: Example of a custom function in a Bash script.....                                                                                       | 29 |
| Figure 71: Output of the above function script.....                                                                                                 | 29 |

## Introduction

This report will be discussing topics relevant to operating systems, including their key functions, use of commands on both Windows and Linux as well as the use of scripts on Windows and Linux.

## Task 1 – Key functions of general Operating Systems

An operating system is a category of system software that handles the management of a computer's hardware and software, such as memory allocation, program execution, input and output, data management and task scheduling [1].

An operating system is composed of multiple components. The main part of the OS is the kernel, which is the underlying core of the operating system. Kernels handle communication between the hardware and software. The main kinds of kernels are Monolithic Kernels, which have better performance and Microkernels, which are more stable. There are also hybrid kernels, which use a mixture of the two technologies for a best of both worlds approach. [2]

General purpose operating systems are what most users will work with. They generally have support for a large amount of system architectures and are often designed with a graphical user interface in mind. Most general purpose operating systems are designed in a way that allows them to run multiple programs at once, often by time-sharing or scheduling programs to run many of them at once [3]. Some systems have multiple CPU – or Central Processing Unit - cores, which will allow the operating system to run programs without needing to schedule different programs [3]. [4]

Most operating systems provide a user interface. Command line interfaces, or CLIs, are the most basic form of user interface and only include text [5]. CLIs are usually interacted with by typing commands on a keyboard [5]. Graphical user interfaces, or GUIs, utilise graphical elements and are usually operated with a mouse or touchscreen [6]. GUIs are considered easier to use but they are more intensive on system resources [7]. This means that operating systems designed for non-user facing devices such as servers will normally use a CLI for performance reasons while devices designed for end users, such as personal computers, will usually have a GUI to make them easier to use [8].

One of the most vital tasks performed by an operating system is memory management. This is usually done by provisioning RAM to specific programs based on what is required. For devices that do not have much physical RAM, such as budget laptops which often come with 4GB of RAM, an operating system may often utilise part of the installed hard drive as virtual memory to allow software to continue running, albeit at a noticeable performance cost. [4]

As a result of increasing requirements for security in the modern day and age, the majority of operating systems have security features built in [9]. For example, a modern operating system may include features such as full drive encryption [10], integrated firewalls and user privilege configuration. It is also common to bundle anti-virus software with an operating system.

Finally, the majority of modern day operating systems are capable of accessing networks to allow for sending and receiving data from other computers. This can include both local area networks as well as the internet. [4]

Operating systems usually include built-in utilities for viewing the overall performance of these operating system functions. For example, htop (pictured below) is a command line utility for Linux that displays active system processes and the current usage of system components [11].

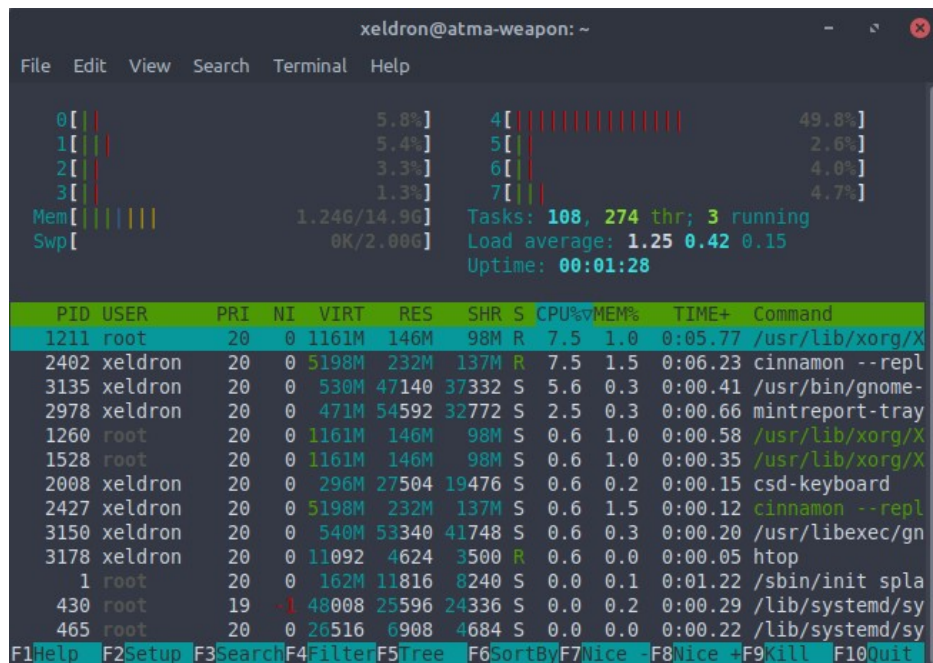


Figure 1: htop utility showing active processes and system resource usage

## Task 2 – Use of Windows OS CLI commands

Windows uses a command line shell called the Command Prompt. Windows also includes a newer, UNIX-inspired command line system called PowerShell which is capable of running all command prompt commands in addition to .NET-based command-lets [12]. Unlike the majority of Linux shells, the Windows command line is not case-sensitive. This means that command, file and folder names do not need to have consistent casing.

- `cd` – changes the current directory [13]

```
C:\Users\benro>cd Pictures
C:\Users\benro\Pictures>
```

Figure 2: Output of the Windows `cd` command

- `echo` – prints a line of text on the screen [14]

```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

C:\Users\benro>echo Hello World
Hello World

C:\Users\benro>
```

Figure 3: Printing "Hello World" with the `echo` command

- `dir` – list the contents of the current directory [15].

```
C:\Users\benro>dir
Volume in drive C has no label.
Volume Serial Number is 74B9-ADE2

Directory of C:\Users\benro

04/05/2023  16:17    <DIR>          .
04/05/2023  16:17    <DIR>          ..
02/07/2022  21:12    <DIR>          3D Objects
02/07/2022  21:12    <DIR>          Contacts
11/08/2022  20:53    <DIR>          Creative Cloud Files
02/07/2022  21:12    <DIR>          Desktop
02/07/2022  22:02    <DIR>          Documents
04/07/2022  22:50    <DIR>          Downloads
02/07/2022  21:12    <DIR>          Favorites
02/07/2022  21:12    <DIR>          Links
02/07/2022  21:12    <DIR>          Music
11/08/2022  20:51    <DIR>          OneDrive
02/07/2022  21:15    <DIR>          Pictures
02/07/2022  21:12    <DIR>          Saved Games
02/07/2022  21:14    <DIR>          Searches
04/07/2022  13:27    <DIR>          Videos
               0 File(s)              0 bytes
               16 Dir(s) 14,553,788,416 bytes free
```

Figure 4: Output of the Windows `dir` command in a user's folder

- mkdir – create a directory. Directories can also be deleted by using rmdir [16], [17].

```
C:\Users\benro>mkdir test

C:\Users\benro>cd test

C:\Users\benro\test>cd ..

C:\Users\benro>rmdir test

C:\Users\benro>cd test
The system cannot find the path specified.
```

Figure 5: Creating a directory with mkdir, moving in and out of it with cd, then deleting it with rmdir

- help – provides a list of commands and information on how to use them [18].

```
C:\Users\benro>help
For more information on a specific command, type HELP command-name
ASSOC      Displays or modifies file extension associations.
ATTRIB     Displays or changes file attributes.
BREAK      Sets or clears extended CTRL+C checking.
BCDEDIT    Sets properties in boot database to control boot loading.
CACLS      Displays or modifies access control lists (ACLs) of files.
CALL       Calls one batch program from another.
CD         Displays the name of or changes the current directory.
CHCP       Displays or sets the active code page number.
CHDIR      Displays the name of or changes the current directory.
CHKDSK     Checks a disk and displays a status report.
CHKNTFS    Displays or modifies the checking of disk at boot time.
CLS        Clears the screen.
CMD        Starts a new instance of the Windows command interpreter.
COLOR      Sets the default console foreground and background colors.
COMP       Compares the contents of two files or sets of files.
COMPACT    Displays or alters the compression of files on NTFS partitions.
CONVERT    Converts FAT volumes to NTFS. You cannot convert the
           current drive.
COPY       Copies one or more files to another location.
DATE       Displays or sets the date.
DEL        Deletes one or more files.
DIR        Displays a list of files and subdirectories in a directory.
DISKPART   Displays or configures Disk Partition properties.
DOSKEY     Edits command lines, recalls Windows commands, and
           creates macros.
DRIVERQUERY Displays current device driver status and properties.
ECHO       Displays messages, or turns command echoing on or off.
ENDLOCAL   Ends localization of environment changes in a batch file.
ERASE      Deletes one or more files.
EXIT       Quits the CMD.EXE program (command interpreter).
FC         Compares two files or sets of files, and displays the
           differences between them.
FIND       Searches for a text string in a file or files.
FINDSTR    Searches for strings in files.
FOR        Runs a specified command for each file in a set of files.
FORMAT     Formats a disk for use with Windows.
FSUTIL     Displays or configures the file system properties.
FTYPE      Displays or modifies file types used in file extension
           associations.
GOTO       Directs the Windows command interpreter to a labeled line in
           a batch program.
GPRESULT   Displays Group Policy information for machine or user.
GRAFTABL   Enables Windows to display an extended character set in
           graphics mode.
HELP       Provides Help information for Windows commands.
ICACLS     Display, modify, backup, or restore ACLs for files and
           directories.
IF         Performs conditional processing in batch programs.
LABEL      Creates, changes, or deletes the volume label of a disk.
MD         Creates a directory.
MKDIR      Creates a directory.
MKLINK     Creates Symbolic Links and Hard Links
MODE       Configures a system device.
```

Figure 6: List of commands produced by the Windows "help" command

```
C:\Users\benro>help echo
Displays messages, or turns command-echoing on or off.

    ECHO [ON | OFF]
    ECHO [message]

Type ECHO without parameters to display the current echo setting.
```

Figure 7: Output of the Windows help command with echo as an argument

- copy, del, move - file management commands. move moves or renames a file, copy copies a file and del deletes a file. The erase command is also an alias for del, and can be used the same way as it [19], [20], [21].

```
C:\Users\benro\copying>copy Thumbs.db Thumbs2.db
1 file(s) copied.

C:\Users\benro\copying>dir
Volume in drive C has no label.
Volume Serial Number is 74B9-ADE2

Directory of C:\Users\benro\copying

17/04/2024  11:39    <DIR>          .
17/04/2024  11:39    <DIR>          ..
17/04/2024  11:20             40,448 Thumbs.db
17/04/2024  11:20             40,448 Thumbs2.db
                2 File(s)              80,896 bytes
                2 Dir(s)  7,332,073,472 bytes free

C:\Users\benro\copying>move Thumbs2.db ThumbsTwo.db
1 file(s) moved.

C:\Users\benro\copying>dir
Volume in drive C has no label.
Volume Serial Number is 74B9-ADE2

Directory of C:\Users\benro\copying

17/04/2024  11:39    <DIR>          .
17/04/2024  11:39    <DIR>          ..
17/04/2024  11:20             40,448 Thumbs.db
17/04/2024  11:20             40,448 ThumbsTwo.db
                2 File(s)              80,896 bytes
                2 Dir(s)  7,332,073,472 bytes free

C:\Users\benro\copying>del ThumbsTwo.db

C:\Users\benro\copying>dir
Volume in drive C has no label.
Volume Serial Number is 74B9-ADE2

Directory of C:\Users\benro\copying

17/04/2024  11:39    <DIR>          .
17/04/2024  11:39    <DIR>          ..
17/04/2024  11:20             40,448 Thumbs.db
                1 File(s)              40,448 bytes
                2 Dir(s)  7,332,114,432 bytes free
```

Figure 8: Performing file management operations on a Thumbs.db file with the Windows copy, del and move commands

- `type` – outputs the contents of a file [22].

```
C:\Users\benro>type LoremIpsum.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
C:\Users\benro>
```

*Figure 9: Outputting dummy text from a plain text file using the Windows type command*

### Task 3 – Use of Linux OS CLI commands

The Bourne Again Shell, or Bash, is a Free and Open Source command line shell for Unix-like operating systems, including Linux and Mac OS.

In these examples, Linux will be used but the syntax is identical regardless of operating system. Unlike Windows, the majority of Unix-like systems are case sensitive which generally means that commands must be typed entirely in lowercase.

- `cd` – changes the current directory [23]

```
cd: usage: cd [-L|[-P [-e]] [-@]] [dir]
xeldron@xeldron-inspiron3480:~$ cd Documents/
xeldron@xeldron-inspiron3480:~/Documents$
```

Figure 10: Changing directories on Linux with the `cd` command

- `ls` – list the contents of the current directory [24].

```
xeldron@xeldron-inspiron3480:~/config/gzdoom$ ls
doom.id.doom1.ultimate  DSD_Remake_Complete.zip      d.WAD      soundfonts
DOOM.WAD                'DSD remake gore edition.pk3' gzdoom.ini
DSD_Remake_Complete    'DSD remake.pk3'            sf2
xeldron@xeldron-inspiron3480:~/config/gzdoom$
```

Figure 11: Listing the contents of a game folder using the Bash `ls` command

- `echo` – prints a line of text on the screen [25].

```
xeldron@xeldron-inspiron3480:~$ echo "Hello World!"
Hello World!
xeldron@xeldron-inspiron3480:~$
```

Figure 12: Outputting "Hello World" with the Bash `echo` command

- `mv`, `cp`, `rm` – file management commands. `mv` moves or renames a file, `cp` copies a file and `rm` deletes a file [26], [27], [28].

```
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ls
example
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ cp example example2
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ls
example  example2
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ mv example demonstration
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ls
demonstration  example2
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ rm example2
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ls
demonstration
xeldron@xeldron-inspiron3480:~/MVCPRMExample$
```

Figure 13: Performing file management on Linux with the Bash `mv`, `cp` and `rm` commands

- `mkdir` – create a directory [16]. Directories can also be deleted by using `rmdir` [29].

```
xeldron@xeldron-inspiron3480:~$ mkdir example-folder
xeldron@xeldron-inspiron3480:~$ cd example-folder/
xeldron@xeldron-inspiron3480:~/example-folder$ cd ..
xeldron@xeldron-inspiron3480:~$ rmdir example-folder/
xeldron@xeldron-inspiron3480:~$ cd example-folder/
bash: cd: example-folder/: No such file or directory
xeldron@xeldron-inspiron3480:~$
```

Figure 14: Creating and deleting a directory with the Bash `mkdir` and `rmdir` commands

- `uname` – prints system information, such as the kernel in use [30].

```
xeldron@xeldron-inspiron3480:~$ uname
Linux
xeldron@xeldron-inspiron3480:~$ uname -a
Linux xeldron-inspiron3480 5.15.0-70-generic #77-Ubuntu SMP Tue Mar 21 14:02:37 UTC 2023
x86_64 x86_64 x86_64 GNU/Linux
xeldron@xeldron-inspiron3480:~$
```

Figure 15: Printing system information with the Bash `uname` command

- `man` – view the manual page about that particular command. It is very useful for trying to see how to use a specific command and its syntax [31].

```
MAN(1)                                Manual pager utils                                MAN(1)

NAME
    man - an interface to the system reference manuals

SYNOPSIS
    man [man options] [[section] page ...] ...
    man -k [apropos options] regexp ...
    man -K [man options] [section] term ...
    man -f [whatis options] page ...
    man -l [man options] file ...
    man -w|-W [man options] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section, if provided, will direct man to look only in that section of the manual. The default action is to search in all of the available sections following a pre-defined order (see DEFAULTS), and to show only the first page found, even if page exists in several sections.

    The table below shows the section numbers of the manual followed by the types of pages they contain.

    1 Executable programs or shell commands
    2 System calls (functions provided by the kernel)
    3 Library calls (functions within program libraries)
    4 Special files (usually found in /dev)
    5 File formats and conventions, e.g. /etc/passwd
    6 Games
    7 Miscellaneous (including macro packages and conventions), e.g. man(7),

Manual page man(1) line 1 (press h for help or q to quit)
```

Figure 16: Checking the manual page for the "man" command using the Bash `man` command

- `cat` – outputs the contents of a file [32]. There's also `tac` which prints the output of the file in reverse [33].

```
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ cat loremispum
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incid
idunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitati
on ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in rep
rehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sin
t occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ █
```

*Figure 17: Outputting the contents of a plain text file containing placeholder text with the Bash `cat` command*

- `chmod` – changes a file's permissions. It can be used to make a file executable, readable and/or writeable by all or specific users. It works by inputting an operation, which is in the format of `<user level permission><operator><file level permission>`. A plus adds a permission while a minus removes said permission. For example:
  - `a+x` will make the file executable as all users
  - `u-w` will make it so that the user cannot edit that particular file
  - `g-r` will prevent a group of users from even accessing a certain file
  - `u+rw` will make the file readable, writeable and executable by the current user.
  - `o-wx` will make it so that no other users can write to or execute the file
 Chmod can also work in octal mode, which uses numbers instead of letters. For example:
  - `777` makes the file readable, writeable and executable by all users
  - `744` makes the file readable, writeable and executable by the current users and makes it read only to all other users.
 In general, symbolic mode (the one that uses letters) is more powerful than octal mode because it can be used to set individual permission bits, unlike octal mode which cannot be used to set individual permission bits. [34]

```
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ./example-script
bash: ./example-script: Permission denied
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ chmod a+x example-script
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ ./example-script
Hello World!
xeldron@xeldron-inspiron3480:~/MVCPRMExample$ █
```

*Figure 18: Making a file executable with the Bash `chmod` command*

## Task 4 – Use of Windows OS scripts

Windows contains a built in scripting language called Batch which can be used to automate tasks [35]. As with the command prompt, Batch scripts are not case sensitive.

All standard Windows commands can be used within batch files. A notable command that can be used here is Echo, which will print a line of text on the screen [14].

```
... hello.bat
1  @Echo off
2  :: used to prevent batch from outputting commands
3
4  Echo Hello World
5  :: Prints text on the screen
6
7  Pause
8  :: Holds until the user inputs a keypress
9
10 |
```

Figure 19: Sample "Hello World" batch script

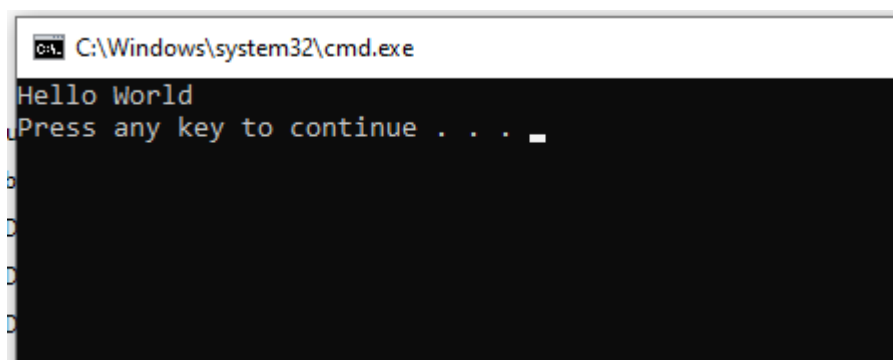


Figure 20: Output of the above hello world script

Variables can be used to store certain information in memory to be accessed and/or modified later [36].

```
... variables.bat
1  @Echo off
2
3  Set var=variable
4  Set /A number=2
5
6  Echo %var%
7  Echo %number%
8  Echo %var% %number%
9
10 pause
11 |
```

Figure 21: Sample batch script utilising variables

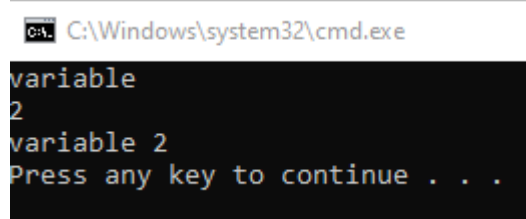
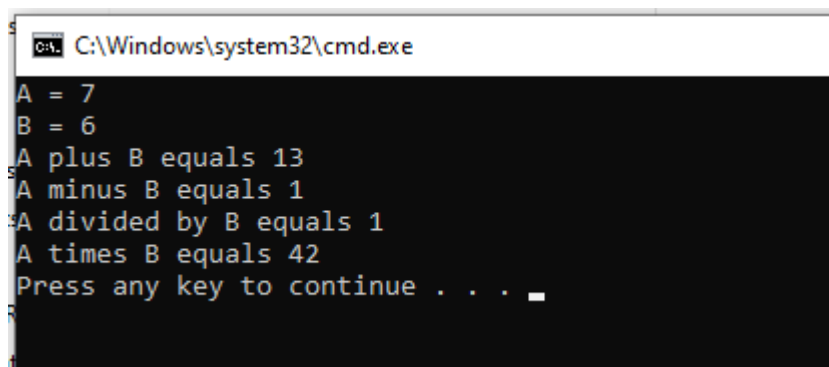


Figure 22: Output of the above variable script

Basic mathematical computations can be performed within Batch by using standard mathematical symbols [37].

```
... maths.bat
1  @Echo off
2
3  set /A a = 7
4  set /A b = 6
5
6  echo A = %a%
7  echo B = %b%
8
9  set /A adding = a + b
10 set /A minus = a - b
11 set /A division = a / b
12 set /A meaningoflife = a * b
13
14 echo A plus B equals %adding%
15 echo A minus B equals %minus%
16 echo A divided by B equals %division%
17 echo A times B equals %meaningoflife%
18
19 pause
20
```

Figure 23: Sample batch file containing arithmetic operations



```
C:\Windows\system32\cmd.exe
A = 7
B = 6
A plus B equals 13
A minus B equals 1
A divided by B equals 1
A times B equals 42
Press any key to continue . . .
```

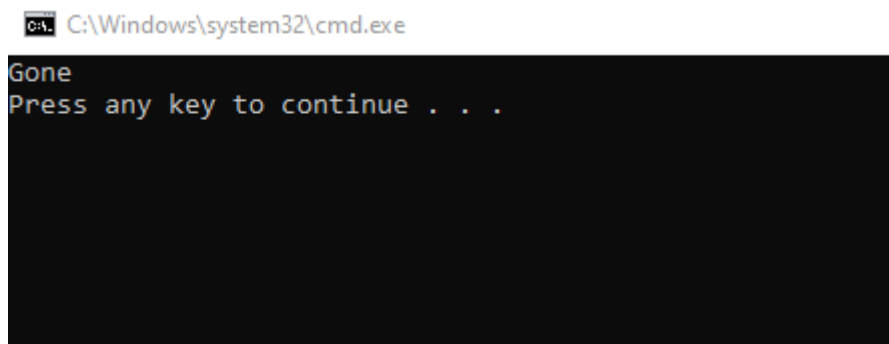
Figure 24: Output of the above arithmetic script

Note that the division operation in the example script does not produce decimal points. This is because batch does not support floating point operations.

If moving to a different part of the script is required, goto commands can be used for this purpose [38].

```
1 @Echo off
2
3 Goto Label
4 > Echo this shouldn't be echoed
5 :Label
6 > Echo Gone
7
8 Pause
9
10
11
```

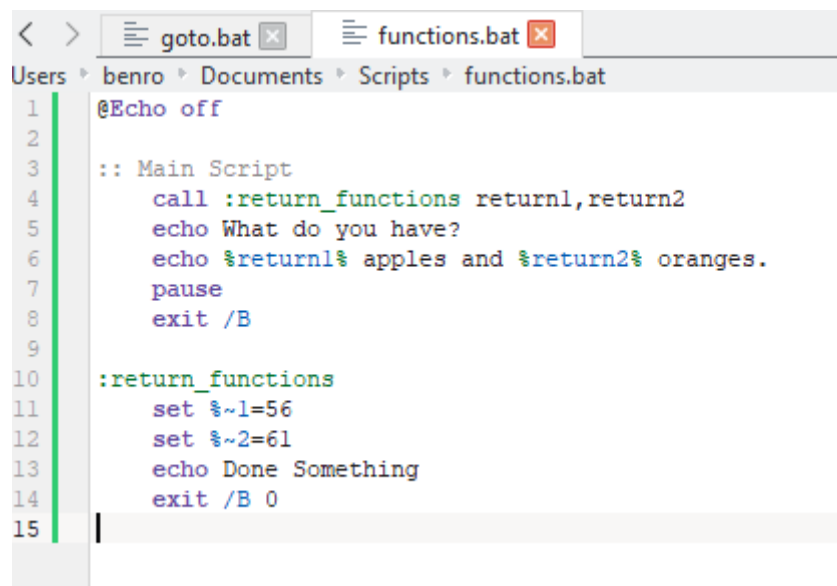
Figure 25: Sample batch script with goto operations



```
C:\Windows\system32\cmd.exe
Gone
Press any key to continue . . .
```

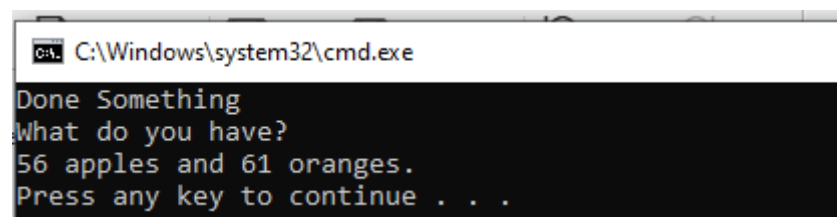
Figure 26: Output of the above script with goto operations

Functions work similarly to goto, but only call the segment rather than jumping to it. This makes them better for larger scripts [39].



```
Users > benro > Documents > Scripts > functions.bat
1 @Echo off
2
3 :: Main Script
4 call :return_functions return1,return2
5 echo What do you have?
6 echo %return1% apples and %return2% oranges.
7 pause
8 exit /B
9
10 :return_functions
11 set %~1=56
12 set %~2=61
13 echo Done Something
14 exit /B 0
15
```

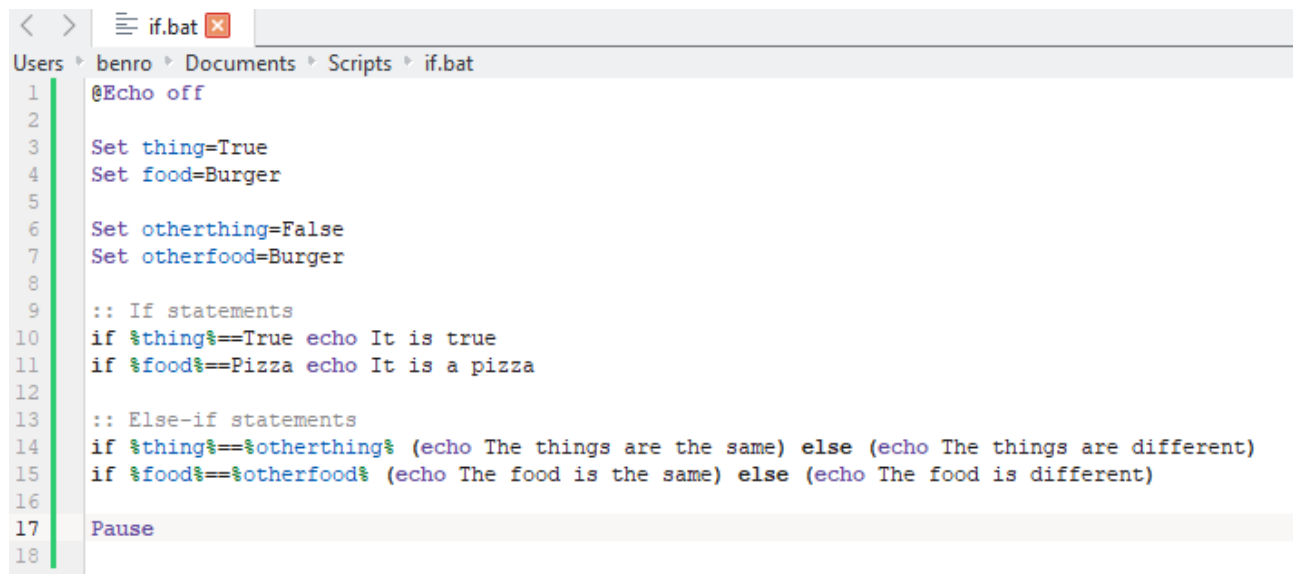
Figure 27: Sample batch script with function operations



```
C:\Windows\system32\cmd.exe
Done Something
What do you have?
56 apples and 61 oranges.
Press any key to continue . . .
```

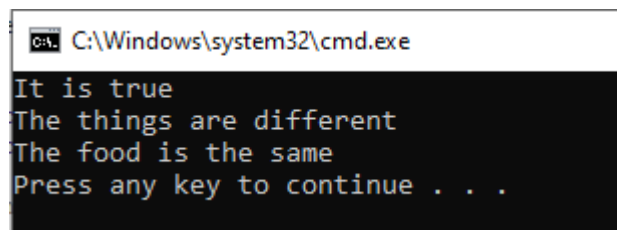
Figure 28: Output of the above function script

If statements can be used to conditionally execute commands based on certain criteria, for example if a variable is equal to a certain value [40].



```
< > if.bat x
Users > benro > Documents > Scripts > if.bat
1 @Echo off
2
3 Set thing=True
4 Set food=Burger
5
6 Set otherthing=False
7 Set otherfood=Burger
8
9 :: If statements
10 if %thing%==True echo It is true
11 if %food%==Pizza echo It is a pizza
12
13 :: Else-if statements
14 if %thing%==%otherthing% (echo The things are the same) else (echo The things are different)
15 if %food%==%otherfood% (echo The food is the same) else (echo The food is different)
16
17 Pause
18
```

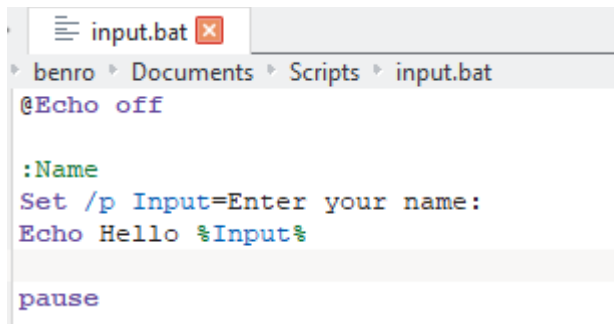
Figure 29: Sample batch script demonstrating if statements



```
C:\Windows\system32\cmd.exe
It is true
The things are different
The food is the same
Press any key to continue . . .
```

Figure 30: Output of the above script with statements

User input can be recorded with the set command and saved to a variable [41].

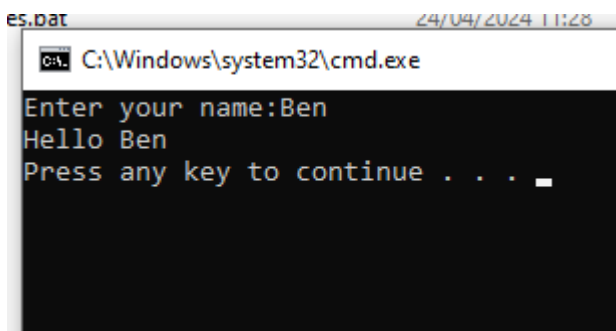


```
input.bat
benro > Documents > Scripts > input.bat
@Echo off

:Name
Set /p Input=Enter your name:
Echo Hello %Input%

pause
```

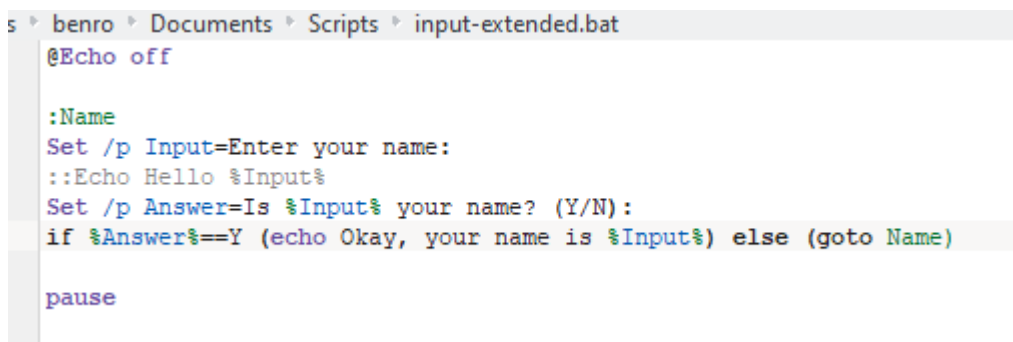
Figure 31: Sample batch script containing a set command to store user input as a



```
es.bat 24/04/2024 11:28
C:\Windows\system32\cmd.exe
Enter your name:Ben
Hello Ben
Press any key to continue . . .
```

Figure 32: Output of the above script, demonstrating user input

User input can easily work with if and goto in order to make a script interactive.

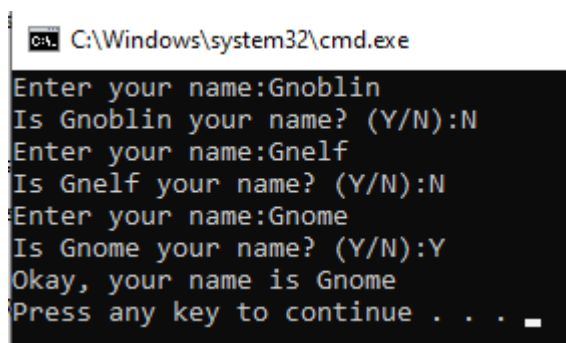


```
benro > Documents > Scripts > input-extended.bat
@Echo off

:Name
Set /p Input=Enter your name:
::Echo Hello %Input%
Set /p Answer=Is %Input% your name? (Y/N):
if %Answer%==Y (echo Okay, your name is %Input%) else (goto Name)

pause
```

Figure 33: Sample batch script using if and goto in order to simulate a dialogue confirmation



```
C:\Windows\system32\cmd.exe
Enter your name:Goblin
Is Goblin your name? (Y/N):N
Enter your name:Gnelf
Is Gnelf your name? (Y/N):N
Enter your name:Gnome
Is Gnome your name? (Y/N):Y
Okay, your name is Gnome
Press any key to continue . . .
```

Figure 34: Output of the above script with user input, returning to an earlier point in the script when the option is denied

An efficient way to run the same code multiple times would be loops, which are also supported in batch. In this example, a for loop is used to call an echo command 12 times, each time showing a different number [42].

```
s > benro > Documents > Scripts > forloop.bat
@Echo off

for %%x in (1 2 3 4 5 6 7 8 9 10 11 12) do (
    echo %%x
)

pause
```

Figure 35: Sample batch script with a for loop

While more efficient than having 12 separate echo commands, the above code is still

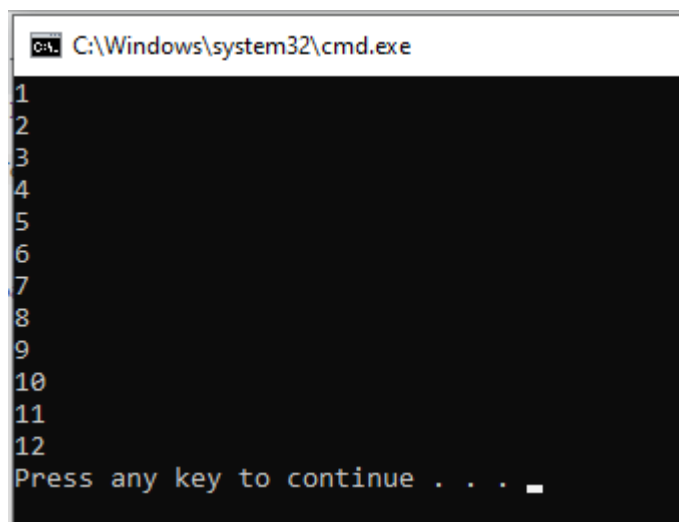


Figure 36: Output of the above for loop

somewhat inefficient. A better way to get the above result would be to use a for loop with iterators

```
> input.bat x input-extended.bat x i
ers > benro > Documents > Scripts > forloop.bat
@Echo off

for /L %%x in (1, 1, 12) do (
    echo %%x
)

pause
```

Figure 37: Improved version of the above script that auto-increments instead of iterating through a list

Arrays can be used to store data in a table, using a number to determine what index in the array to write to or read to [43].

```
@Echo off

Set TheArray[0]=Eggs
Set TheArray[1]=Spam
Set TheArray[2]=Ham

echo %TheArray[0]%
echo %TheArray[1]%
echo %TheArray[2]%

echo %TheArray[3]%
:: Should print the current echo state because the entry is blank

:: Change Spam to Baked Beans
Set TheArray[1]=Baked Beans

echo %TheArray[1]%

pause
```

Figure 39: Sample batch script showing reading and writing from and to an array

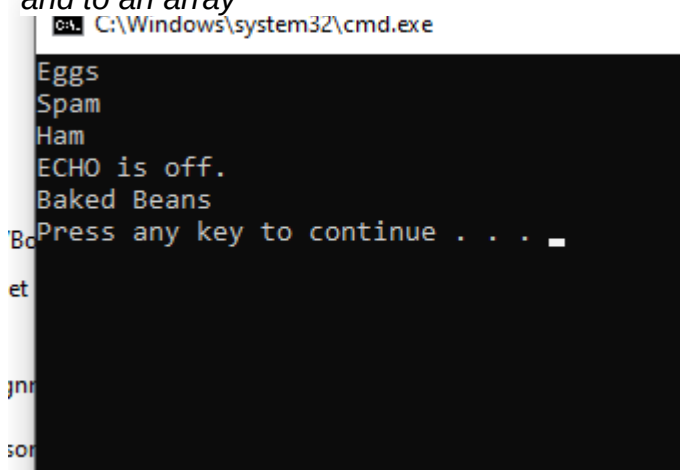


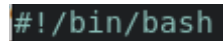
Figure 38: Output of the above array script

It is not possible to iterate through an array using a loop in batch.

Batch is well suited to system automation, but would not be recommended for complex programs due to its limitations.

## Task 5 – Use of Linux OS scripts

Linux is also able to run commands from a script file. Linux scripts are normally started with a shebang, which tells the file what interpreter to use. For example,

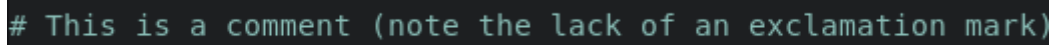


```
#!/bin/bash
```

*Figure 40:  
POSIX  
shebang  
pointing a  
script to the  
Bash shell  
interpreter*

tells the system to use the Bash shell. This is important many users use different CLI shells to Bash, and these shells are often incompatible [44].

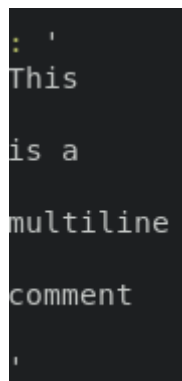
Comments can be inserted into Linux scripts using a hash symbol (#). For example,



```
# This is a comment (note the lack of an exclamation mark)
```

*Figure 41: Single-line Bash shell script comment prefixed with a hash symbol*

would be ignored by the script. Multiline comments are can also be done like this:



```
: '  
This  
is a  
multiline  
comment  
'
```

*Figure 42:  
Bash shell  
script  
multiline  
comment*

Alternatively, this method of commenting also works:

```
<<COMMENTS  
  
This  
is  
also  
a  
multiline  
comment  
  
COMMENTS
```

*Figure 43:  
Alternate  
method of  
adding a  
multiline  
comment in a  
Bash shell script*

In addition to having support for all standard Linux commands such as echo, general scripting functionality is supported; not unlike a programming language.

In Bash, a variable is set up similarly to how It would be in most programming languages, but no spaces are allowed. They can be read by prefixing the name of a variable with a \$ sign [45].

```
exampleVariable="potatoes"  
  
echo $exampleVariable # This prints potatoes
```

*Figure 44: Sample Bash shell script demonstrating the  
use of variables with an echo command*

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./echovar.sh  
potatoes  
xeldron@atma-weapon:~/Downloads/Scripts$
```

*Figure 45: Output of the above variable script*

There is also a command that can be used to set variables based on user input; specifically the read command.

```
read -p "Pick a number: ", numberPicked  
  
echo $numberPicked # Prints the value that the user entered
```

*Figure 46: Sample shell script to store and output user input*

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./read.sh  
Pick a number: 6  
6  
xeldron@atma-weapon:~/Downloads/Scripts$
```

*Figure 47: Output of the above input script*

Arithmetic in Bash uses the `let` and `expr` commands for arithmetic. `let` saves the value to a variable, while `expr` just echoes the result [46].

```
#!/bin/bash

let num=1+2
echo $num

let ber=4*5

expr 2 + 2
```

Figure 48: Sample bash script for performing

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./expr.sh
3
4
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 49: Output of the above arithmetic script

If Statements can be used in bash with the `if` command. An if statement is ended with the `fi` command. [46]

```
exampleVariable="potatoes"

echo $exampleVariable # This prints potatoes

if [ $exampleVariable == "tomatoes" ]
then
    echo "There are tomatoes"
else
    echo "There aren't any tomatoes"
fi
```

Figure 50: Shell script using an if statement to run different commands based on a variable

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./variables.sh
potatoes
There aren't any tomatoes
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 51: Output of the above script with if statements

Else-if statements are also supported with elif [46]:

```
#!/bin/bash

score=8

if [ $score -lt 4 ] || [ $score -eq 4 ] ;
then
    echo "Ha ha you're a loser"
elif [ $score -gt 5 ] && [ $score -lt 9 ] || [ $score -eq 9 ] || [ $score -eq 5 ] ;
then
    echo "A winner is you"
else
    echo "A cheat, a cheat, that's what you are!"
fi
```

Figure 52: Bash script showing the use of less than and greater than operations as well as elif to cycle through responses based on a variable

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./if.sh
A winner is you
```

Figure 53: Output of the above shell script showing the result with a value of "8"

Case statements are similar to if statements, but they allow you to include a variety of answers under the same result. This makes them useful for system prompts [46].

```
#!/bin/bash

echo "Do you believe in ghosts?"
read -p "Yes/No :" Answer

case $Answer in
    Yes|yes|Y|y)
        echo "So some people really do believe"
        ;;
    No|no|N|n)
        echo "Then what about the hand on your shoulder?"
        ;;
    *)
        echo "Spooky."
        ;;
esac
```

Figure 54: Batch Script utilising case switching

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./caseswitch.sh
Do you believe in ghosts?
Yes/No :Yes
So some people really do believe
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 55: Output of the above switch utilising case switching

Loops run the same command multiple times. Bash supports three kinds of loops. For Loops run a set number of times [46].

```
for counter in {1..10..1}
do
    echo $counter
done
```

Figure 56: Simple Bash shell script making use of a for loop

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./forcounter.sh
1
2
3
4
5
6
7
8
9
10
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 57: Output of the above for loop script

While Loops run until the condition is false [46]

```
#!/bin/bash

read -p "Enter the starting number: " firstNumber
read -p "Enter the final number: " lastNumber

while [[ $firstNumber -le $lastNumber ]];
do
    echo $firstNumber
    ((firstNumber++))
done
```

Figure 58: Example Bash shell script with a while loop

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./whileloop.sh
Enter the starting number: 1
Enter the final number: 5
1
2
3
4
5
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 59: Output of the above while loop script

Until Loops run until the condition is true [46]

```
#!/usr/bin

User_input=null

until [ "$User_input" = "The Password"
      echo "Enter The Password"
      read User_input
      if [ $User_input != "The Password" ];
        echo "That was not The Password"
      fi
done

echo "You entered the password"
```

Figure 60: Bash shell script containing an until loop

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./inputuntil.sh
Enter The Password
Swordfish
That was not The Password
Enter The Password
The Password
You entered The Password
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 61: Output of the above until loop script

Bash scripting can also apply ANSI formatting to the echo command. This can be used to recolour text. For example,

```
echo -e "\e[1;34m This Text Is Blue"
```

Figure 62: Bash echo command in a script being ran with the -e flag, enabling the use of escape characters for text formatting

produces

```
This Text Is Blue
```

Figure 63: Output of the above command

ANSO codes can also be used in echo commands, allowing for additional text formatting:

|                                       |              |
|---------------------------------------|--------------|
| echo -e "\e[1mBold Text\e[0m"         | Bold Text    |
| echo -e "\e[3mItalics\e[0m"           | Italics      |
| echo -e "\e[3m\e[1mBold Italics\e[0m" | Bold Italics |
| echo -e "\e[4mUnderline\e[0m"         | Underline    |
| echo -e "\e[31mRed Text\e[0m"         | Red Text     |

Figure 64: A series of Bash echo commands with ANSO escape codes

Figure 65: Output of the above formatting script

ANSI codes are also supported, but additional functions need to be set up in the script to work:

```
ansi() { echo -e "\e[${1}m${*:2}\e[0m"; }

bold() { ansi 1 "$@"; }
italic() { ansi 3 "$@"; }
underline() { ansi 4 "$@"; }
strikethrough() { ansi 9 "$@"; }
red() { ansi 31 "$@"; }

echo "$(bold Bold Text)"
echo "$(italic Italic Text)"
echo "$(underline Underlined Text)"
echo "$(strikethrough Text that's been crossed out)"
echo "$(red Red Text Text)"
```

Figure 66: Bash script file utilising functions to enable text formatting with ANSI codes

```
Bold Text
Italic Text
Underlined Text
Text that has been crossed out
Red Text
```

Figure 67: Output of the above script with ANSI formatting

Bash also supports list arrays.

```
#!/bin/bash

# Method 1 - Single-line Array

declare singleLineArray=( "Single" "Line" "Array" )

echo ${singleLineArray[0]}
echo ${singleLineArray[@]}

#Method 2 - Multiline Array

declare -a multiLineArray = (
    [1]="Multiline"
    [2]="Array"
)

echo ${multiLineArray[1]}

#Method 3 - Multiline Associative Array

declare -A multilineAssociativeArray=(
    [KindOfArray]="Multiline Associative"
    [Breakfast]="Weetabix"
    [Colnel]="Sanders"
    [WhatAmIFightingFor]="ERROR 404 - Zero Not Found"
)

echo ${multilineAssociativeArray[KindOfArray]}
```

Figure 68: Bash script displaying a variety of implementations for Arrays in

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./array.sh
Single
Single Line Array
Multiline
Multiline Associative
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 69: Output of the above array script

Finally, Bash can also utilise custom functions that can be called throughout the script.

```
Funky() {
    echo "This is a function"
}

Funky
```

Figure 70: Example of a custom function in a Bash script

```
xeldron@atma-weapon:~/Downloads/Scripts$ ./function.sh
This is a function
xeldron@atma-weapon:~/Downloads/Scripts$
```

Figure 71: Output of the above function script

## Conclusion

In conclusion, this document has discussed the general functions of an operating system as well as using the command line on both Windows and Linux/UNIX systems along with shell scripting for both operating systems.

## Bibliography

- [1] Merriam-Webster Dictionary, 'Definition of OPERATING SYSTEM'. Accessed: May 08, 2024. [Online]. Available: <https://www.merriam-webster.com/dictionary/operating+system>
- [2] Codecademy, 'What Is a Kernel?', Codecademy Blog. Accessed: Apr. 29, 2024. [Online]. Available: <https://www.codecademy.com/resources/blog/kernel/>
- [3] Y. Wiseman and S. Jiang, *Advanced Operating Systems and Kernel Applications: Techniques and Technologies*. IGI Global, 1AD. Accessed: May 14, 2024. [Online]. Available: <https://www.igi.global.com/gateway/book/11>
- [4] Bos Herbert and Tanenbaum Andrew S, *Modern operating systems*, Global edition, Fourth edition. in Always Learning. Harlow, England: Pearson, 2014.
- [5] 'The Linux command line for beginners', Ubuntu. Accessed: May 08, 2024. [Online]. Available: <https://ubuntu.com/tutorials/command-line-for-beginners>
- [6] B. J. Jansen, 'The graphical user interface', *SIGCHI Bull.*, vol. 30, no. 2, pp. 22–26, Apr. 1998, doi: 10.1145/279044.279051.
- [7] W. L. Martinez, 'Graphical user interfaces', *WIREs Computational Statistics*, vol. 3, no. 2, pp. 119–133, 2011, doi: 10.1002/wics.150.
- [8] D. G, 'What Is CLI?', Hostinger Tutorials. Accessed: May 08, 2024. [Online]. Available: <https://www.hostinger.com/tutorials/what-is-cli>
- [9] P. Reiher, 'Introduction to Operating System Security'.
- [10] B. Dickens III, H. S. Gunawi, A. J. Feldman, and H. Hoffmann, 'StrongBox: Confidentiality, Integrity, and Performance using Stream Ciphers for Full Drive Encryption', *SIGPLAN Not.*, vol. 53, no. 2, pp. 708–721, Mar. 2018, doi: 10.1145/3296957.3173183.
- [11] 'htop - an interactive process viewer'. Accessed: May 15, 2024. [Online]. Available: <https://htop.dev/>
- [12] sdwheeler, 'What is PowerShell? - PowerShell'. Accessed: May 20, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.4>
- [13] JasonGerend, 'cd'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cd>
- [14] Xelu86, 'echo'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/echo>
- [15] JasonGerend, 'dir'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/dir>
- [16] JasonGerend, 'mkdir'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/mkdir>
- [17] JasonGerend, 'rmdir'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/rmdir>
- [18] JasonGerend, 'help'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/help>

- [19] JasonGerend, 'copy'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/copy>
- [20] JasonGerend, 'del'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/del>
- [21] JasonGerend, 'move'. Accessed: May 20, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/move>
- [22] JasonGerend, 'type'. Accessed: May 15, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/type>
- [23] 'cd(1p) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://www.man7.org/linux/man-pages/man1/cd.1p.html>
- [24] 'ls(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/ls.1.html>
- [25] 'echo(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/echo.1.html>
- [26] 'mv(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/mv.1.html>
- [27] 'cp(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/cp.1.html>
- [28] 'rm(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/rm.1.html>
- [29] 'rmdir(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/rmdir.1.html>
- [30] 'uname(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/uname.1.html>
- [31] 'man(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/man.1.html>
- [32] 'cat(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/cat.1.html>
- [33] 'tac(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/tac.1.html>
- [34] 'chmod(1) - Linux manual page'. Accessed: May 20, 2024. [Online]. Available: <https://man7.org/linux/man-pages/man1/chmod.1.html>
- [35] 'Using batch files'. Accessed: May 20, 2024. [Online]. Available: [https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490869\(v=technet.10\)](https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490869(v=technet.10))
- [36] 'Batch Script - Variables'. Accessed: May 28, 2024. [Online]. Available: [https://www.tutorialspoint.com/batch\\_script/batch\\_script\\_variables.htm](https://www.tutorialspoint.com/batch_script/batch_script_variables.htm)
- [37] J. W. S. Parker, 'How can I perform arithmetic operations in a Batch file?', James Parker. Accessed: May 28, 2024. [Online]. Available: <https://www.jamesparker.dev/how-can-i-perform-arithmetic-operations-in-a-batch-file/>
- [38] JasonGerend, 'goto'. Accessed: May 28, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/goto>
- [39] 'Batch File Functions (Introduction And Tutorial)', Online tutorials for c programming, cplusplus, Java, Python. Accessed: May 28, 2024. [Online]. Available: <https://www.trytoprogram.com/batch-file-functions/>

- [40] JasonGerend, 'if'. Accessed: May 28, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/if>
- [41] 'Batch Script - Input / Output', GeeksforGeeks. Accessed: May 28, 2024. [Online]. Available: <https://www.geeksforgeeks.org/batch-script-input-output/>
- [42] M. A. Islam, 'The For Loop in Batch Script', Delft Stack. Accessed: May 28, 2024. [Online]. Available: <https://www.delftstack.com/howto/batch/batch-for-loop/>
- [43] 'Batch Script - Arrays'. Accessed: May 28, 2024. [Online]. Available: [https://www.tutorialspoint.com/batch\\_script/batch\\_script\\_arrays.htm](https://www.tutorialspoint.com/batch_script/batch_script_arrays.htm)
- [44] 'What is Shebang in Linux Shell Scripting?' Accessed: May 29, 2024. [Online]. Available: <https://linuxhandbook.com/shebang/>
- [45] C. Negus, *Linux Bible*. New York, UNITED STATES: John Wiley & Sons, Incorporated, 2015. Accessed: May 28, 2024. [Online]. Available: <http://ebookcentral.proquest.com/lib/trinity/detail.action?docID=1895205>
- [46] E. Siever, S. Figgins, R. Love, and A. Robbins, *Linux in a Nutshell: A Desktop Quick Reference*, Sixth edition. Beijing Köln: O'Reilly Media, 2009.
- [47] K. M. Al-Aubidy, 'Real-Time Operating Systems'.
- [48] 'Why RTOS and What is RTOS?', FreeRTOS. Accessed: Jun. 06, 2024. [Online]. Available: <https://www.freertos.org/about-RTOS.html>
- [49] 'Understanding Cloud Operating Systems: An In-Depth Guide - [Updated May 2024 ]'. Accessed: Jun. 06, 2024. [Online]. Available: <https://digitalgadgetwave.com/understanding-cloud-operating-systems-an-in-depth/>
- [50] 'What is a qubit? | Institute for Quantum Computing'. Accessed: Jun. 06, 2024. [Online]. Available: <https://uwaterloo.ca/institute-for-quantum-computing/quantum-101/quantum-information-science-and-technology/what-qubit>

## Appendices

### Appendix A – OS Tutorial 1

#### General Purpose

General purpose operating systems are what most users will work with. They generally have support for a large amount of system architectures and are often designed with a graphical user interface in mind. Most general purpose operating systems are designed in a way that allows them to run multiple programs at once, often by time-sharing or scheduling programs to run many of them at once [3].

#### Real Time

Real-time operating systems are designed for situations where a device must operate in real time [47]. These can include microcontrollers and embedded devices powering hardware such as vehicles, industrial machinery, healthcare equipment, automated weapons systems, etc. These operating systems are often customised for the specific embedded device that they are powering. Real-time operating systems are designed for the lowest possible response times as lag can potentially cause issues in the heat of the moment. Since many real-time operating systems are meant for microcontrollers, which are very small computers with very low specifications, a lot of real-time time operating systems are designed to be as lightweight as possible. [48]

#### Cloud

A cloud operating system is designed to be permanently connected to the internet and interacting with remote clients. Additionally, a cloud operating system may frequently be designed to run within a hypervisor rather than on bare metal. This means that a cloud OS will either have virtualisation capabilities built in to account for different potential applications, or be lightweight enough to run inside a virtual machine without being overly intensive on the host system's resources. [49]

#### Quantum

Quantum operating systems are operating systems that are designed for use in quantum computing. Unlike traditional computing (which works on binary integers of ones and zeroes), a quantum computer utilises qubits (quantum bits) which in addition to being able to work in a "1" or "0" state can also exist in an intermediate state between the two [50]. Therefore, a quantum computer operating system has to be designed to function with qubits in mind.

## Appendix B – Certificate for Linux Essentials course



Dear Ben Roberts,

Congratulations on completing the NDG Linux Unhatched course in the Cisco Networking Academy. This letter documents you have successfully completed the NDG Linux Unhatched course, which provides an introduction to the Linux command line. Linux is everywhere! As the reach of Linux continues to grow, knowledge of Linux is a core skill for all IT professionals. By completing this course, you have gained a better understanding of Linux.

If you decide to pursue additional knowledge of Linux consider:

### **NDG LINUX ESSENTIALS**

This course is the perfect next step for beginners looking to expand their skills and knowledge of Linux. This full-semester course can be delivered as instructor-led training or as a self-paced learning experience. The NDG Linux Essentials course is designed to prepare you for the Linux Professional Institute Linux Essentials Professional Development Certificate.

Again, congratulations and we wish you continued success!

*Sincerely,  
The NDG Team*

### **NDG LINUX SERIES**

This certification level series offers beginners more rigorous in-depth coverage. The NDG Introduction to Linux I and NDG Introduction to Linux II courses focus on the basic Linux system administration skills needed in preparation for the Linux Professional Institute LPIC-1 certification.

Date 6 Mar 2024