



OPERATING SYSTEMS

Computer Architecture and Operating Systems –
Assignment 1

Danny Thomas (2004365)

Contents

Key functions of general Operating Systems	2
Windows OS CLI Commands	3
Linux OS CLI Commands.....	7
Windows OS Scripts	11
Linux OS Scripts.....	15
Appendices.....	19

Key functions of general Operating Systems

The operating system of a computer can be described as the firmware that manages all the computer's processes, it is made up of the kernel, that controls system hardware and manages applications, the user interface which is the display or graphics that allows the user to interact with the computer, lastly the API which is where application developers write their code [1].

A key component in a computer is the processor and its memory, therefore memory and processor management are vital for the operating system to handle, the processor management part of the OS manages what processes have access to the CPU and how much time it has to process it, it also has a function called the "traffic controller" which tracks the status of the processes [2]. The operating system handles the memory by allocating bytes of memory to the processes that need it, this is important as memory addressing is used to store data correctly and then fetch it again when its required.

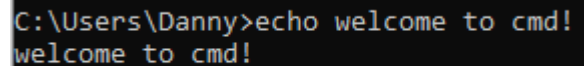
The OS also manages applications that run on a computer, this is used to track how often applications are used and how much memory is allocated to them so that applications can run efficiently for the user.

One of the most important parts of a computer operating system is how they hand the I/O, or the Input/ Output system; as without this you would not be able to plug a keyboard and mouse into your pc and use it, rendering it useless. Of course this handles all external devices such as USB's, Printers, Gamepad controllers and anything else you can use to interact with a computer.

Windows OS CLI Commands

For this task I used Command Prompt, CMD comes installed on every Windows operating system and I use Windows 10 at home, so I was able to use it on my own system. To complete this task I used phoenixsnap.com [3] and digitalcitizen [4] which are both websites with a long list of commands that I got my information from about the different commands such as what they do and how to use them in CMD.

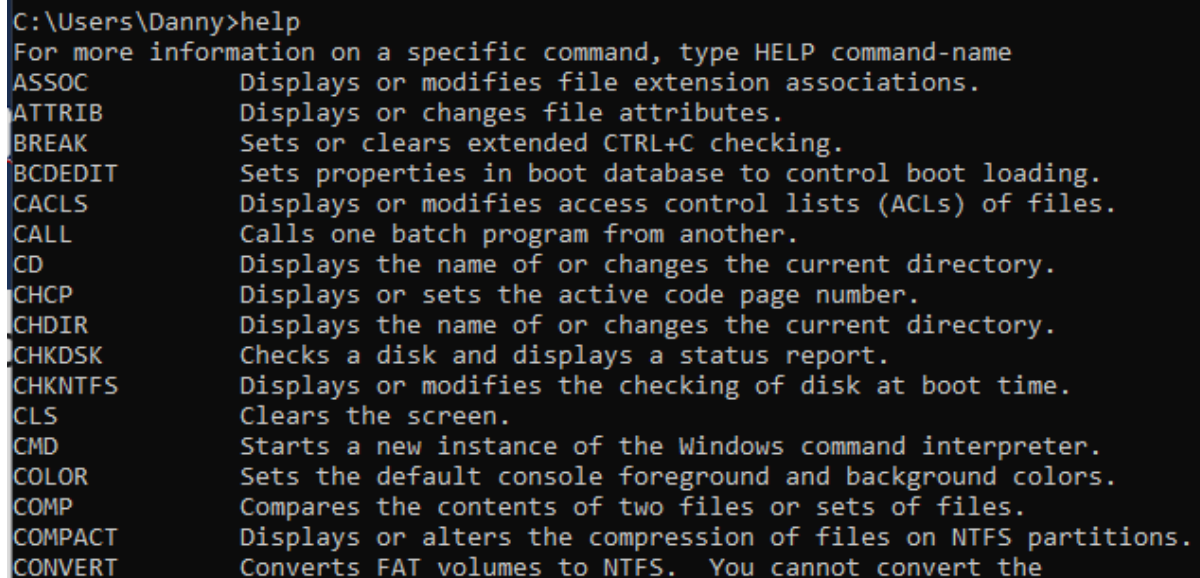
The first command I used was the *echo* command, this simply outputs text into CMD for the user to read.



```
C:\Users\Danny>echo welcome to cmd!  
welcome to cmd!
```

Figure 1: echo command

Secondly was the *help* command, this is perfect for starting out in CMD as it lists many commands and gives a short description of what it does, below is only a small screenshot of the full list.



```
C:\Users\Danny>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
```

Figure 2: help command

Thirdly is the clear command, used by typing *cls*. This will clear CMD and all your previous commands.



```
Command Prompt - cmd  
C:\Users\Danny>cls
```

Figure 3: clear command

```
C:\Users\Danny>whoami
danny-pc\danny
```

The fourth command is *whoami*, this will display your pc name and current account name.

Figure 4: *whoami* command

The fifth command is *tasklist*, this is a very useful command as it shows the user all the currently running processes on the pc, the first 2 columns are the unique identifiers for the different processes, and the column labelled memory usage denotes how much memory that process is using.

```
C:\Users\Danny>tasklist
```

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0	Services	0	8 K
System	4	Services	0	2,644 K
Registry	148	Services	0	91,108 K
smss.exe	508	Services	0	1,060 K
csrss.exe	688	Services	0	5,908 K
wininit.exe	904	Services	0	6,920 K
csrss.exe	912	Console	1	6,348 K
services.exe	976	Services	0	14,556 K
lsass.exe	984	Services	0	22,988 K
svchost.exe	812	Services	0	29,032 K
WUDFHost.exe	852	Services	0	6,880 K
fontdrvhost.exe	864	Services	0	3,304 K
svchost.exe	1044	Services	0	17,684 K
svchost.exe	1100	Services	0	8,096 K
winlogon.exe	1168	Console	1	12,144 K
fontdrvhost.exe	1228	Console	1	14,968 K
dwm.exe	1296	Console	1	112,364 K
svchost.exe	1336	Services	0	7,624 K
svchost.exe	1344	Services	0	11,296 K
svchost.exe	1360	Services	0	11,756 K

Figure 5: *tasklist* command

The sixth command is *start*; this can be used to start any program by typing in the program name.

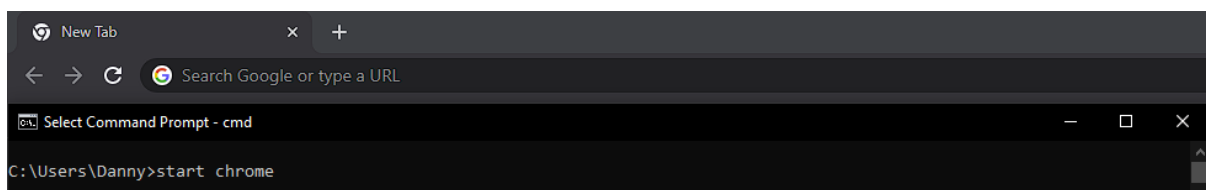


Figure 6: *start* command

Next is the *label* command, label will display the type of the current disk or a specified disk, and its serial number. You can also change its name but note that this requires admin permissions.

```
C:\Users\Danny>label
Volume in drive C: is SSD
Volume Serial Number is 52C5-D953
Volume label (32 characters, ENTER for none)?
```

Figure 7: *label* command

The next command is *dir*, *cd* and *mkdir*. These are commonly used commands that are used to list the computers directories, navigate them, and make new directories respectively.

```
C:\Users\Danny>dir
Volume in drive C is SSD
Volume Serial Number is 52C5-D953

Directory of C:\Users\Danny

25/01/2023  03:09    <DIR>          .
25/01/2023  03:09    <DIR>          ..
02/01/2022  18:07    <DIR>          .android
19/11/2021  22:24    <DIR>          .cache
12/10/2021  16:45    <DIR>          .conda
29/03/2023  17:12    <DIR>          .dwagent
12/10/2021  11:36    <DIR>          .idlerc
12/10/2021  17:07    <DIR>          .ipython
12/10/2021  17:08    <DIR>          .jupyter
12/10/2021  17:34    <DIR>          .matplotlib
11/11/2022  15:02    176 .packettracer
10/10/2021  15:06    <DIR>          .thumbnails
10/02/2022  17:13    <DIR>          .vscode
```

Figure 8: dir command

```
C:\Users\Danny>cd desktop
C:\Users\Danny\Desktop>mkdir folder
```

Figure 9: cd command



Figure 8: mkdir command

Lastly is the *color* command, this command will allow you to edit the color of CMD and will also give you instructions on how to do it.

```
C:\Users\Danny>color 0a
C:\Users\Danny>echo green text!
green text!
```

Figure 101: color command

```
C:\Users\Danny>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
```

Figure 92: color command instructions

Linux OS CLI Commands

To complete this task I installed Linux on a spare laptop and created a new user, all of the screenshots I took were taken of the terminal where the commands were ran.

The first screenshot I took was of the *help* command, similarly to windows CMD this outputted a bunch of common commands and how to use them.

```

user@Lenovo-ideapad-330S-15ARR: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

user@Lenovo-ideapad-330S-15ARR:~$ help
GNU bash, version 5.2.2(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally. Type 'help' to see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

job_spec [&]
(( expression ))
. filename [arguments]
:
[ arg... ]
[[ expression ]]
alias [-p] [name=value] ... ]
bg [job_spec ...]
bind [-lpsvPSVX] [-m keymap] [-f file]
break [n]
builtin [shell-builtin [arg ...]]
caller [expr]
case WORD in [PATTERN] [PATTERN]...)>
cd [-L|[-P [-e]] [-@]] [dir]
command [-pVv] command [arg ...]
compgen [-abcdefgjkuv] [-o option] [>
complete [-abcdefgjkuv] [-pr] [-DEI]>
compopt [-o|+o option] [-DEI] [name .>
continue [n]
coproc [NAME] command [redirections]
declare [-aAfFiIlNrtux] [name=value>
dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [job_spec ... | pid >
echo [-neE] [arg ...]
enable [-a] [-dnps] [-f filename] [na>
eval [arg ...]
exec [-cl] [-a name] [command [argume>
exit [n]
export [-fn] [name=value] ...] or ex>
false
fc [-e ename] [-lnr] [first] [last] o>
fg [job_spec]
for NAME [in WORDS ... ] ; do COMMAND>
for (( exp1; exp2; exp3 )); do COMMAN>
function name { COMMANDS ; } or name >
getopts optstring name [arg ...]
hash [-lr] [-p pathname] [-dt] [name >
help [-dms] [pattern ...]
history [-c] [-d offset] [n] or hist>
if COMMANDS; then COMMANDS; [ elif C>
jobs [-lnprs] [jobspec ...] or jobs >
kill [-s sigspec | -n signum | -sigs>
let arg [arg ...]
local [option] name[=value] ...
logout [n]
mapfile [-d delim] [-n count] [-O or>
popd [-n] [+N | -N]
printf [-v var] format [arguments]
pushd [-n] [+N | -N | dir]
pwd [-LP]
read [-ers] [-a array] [-d delim] [->
readarray [-d delim] [-n count] [-O >
readonly [-aAf] [name[=value] ...] o>
return [n]
select NAME [in WORDS ... ;] do COMM>
set [-abefhkmnptuvxBCEHPT] [-o optio>
shift [n]
shopt [-pqsu] [-o] [optname ...]
source filename [arguments]
suspend [-f]
test [expr]
time [-p] pipeline
times
trap [-lp] [[arg] signal_spec ...]
true
type [-afptP] name [name ...]
typeset [-aAfFiIlNrtux] name[=value>
ulimit [-SHabcdefiklmnpqrstuvxPRT] [>
umask [-p] [-S] [mode]
unalias [-a] name [name ...]
unset [-f] [-v] [-n] [name ...]
until COMMANDS; do COMMANDS-2; done
variables - Names and meanings of so>
wait [-fn] [-p var] [id ...]
while COMMANDS; do COMMANDS-2; done
{ COMMANDS ; }
user@Lenovo-ideapad-330S-15ARR:~$

```

The next command I ran was *date* which displayed the current date, *whoami* which displays the name of the current user (my user name was set to “user”) and *w* which displays who is currently online, and because there’s only one account on my laptop, only the details for “user” appear.

```
user@Lenovo-ideapad-330S-15ARR:~$ date
Thu 6 Apr 14:52:24 BST 2023
user@Lenovo-ideapad-330S-15ARR:~$ whoami
user
user@Lenovo-ideapad-330S-15ARR:~$ w
14:52:42 up 11 min,  1 user,  load average: 0.01, 0.13, 0.13
USER      TTY      FROM          LOGIN@   IDLE   JCPU   PCPU   WHAT
user      tty2     tty2          14:43    11:11  0.03s  0.03s  /usr/libexec/gnome-session-binary --session=ubu
```

Next was *ls* and *ls -R* which listed all directories and all sub directories respectively.

```
user@Lenovo-ideapad-330S-15ARR: ~
user@Lenovo-ideapad-330S-15ARR:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
user@Lenovo-ideapad-330S-15ARR:~$ ls -R
.:
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos

./Desktop:

./Documents:

./Downloads:

./Music:

./Pictures:
Screenshots

./Pictures/Screenshots:
'Screenshot from 2023-04-06 14-44-29.png'  'Screenshot from 2023-04-06 14-53-26.png'

./Public:

./snap:
firefox  snapd-desktop-integration

./snap/firefox:
2487  common  current

./snap/firefox/2487:

./snap/firefox/common:

./snap/snapd-desktop-integration:
14  57  common  current

./snap/snapd-desktop-integration/14:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

./snap/snapd-desktop-integration/14/Desktop:

./snap/snapd-desktop-integration/14/Documents:

./snap/snapd-desktop-integration/14/Downloads:

./snap/snapd-desktop-integration/14/Music:

./snap/snapd-desktop-integration/14/Pictures:

./snap/snapd-desktop-integration/14/Public:

./snap/snapd-desktop-integration/14/Templates:
```

The next command I used was *uptime* which simply outputs how long the laptop has been awake for, after that is the *uname -a* which shows system info and the kernel which is useful to know when updating firmware. I also used the *mount* command which shows the filesystem.

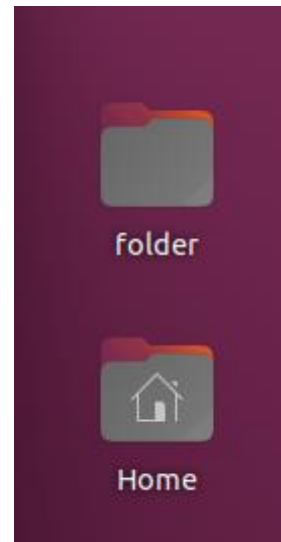
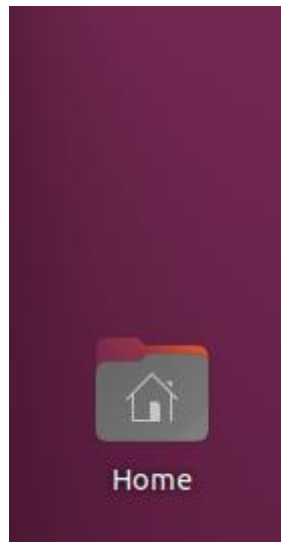
```

user@Lenovo-Ideapad-330S-15ARR: ~
user@Lenovo-ideapad-330S-15ARR:~$ uptime
14:56:19 up 14 min, 1 user, load average: 0.18, 0.15, 0.13
user@Lenovo-ideapad-330S-15ARR:~$ uname -a
Linux Lenovo-ideapad-330S-15ARR 5.19.0-38-generic #39-Ubuntu SMP PREEMPT_DYNAMIC Fri Mar 17 17:33:16 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
user@Lenovo-ideapad-330S-15ARR:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=3461976k,nr_inodes=865494,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=699940k,mode=755,inode64)
/dev/nvme0n1p5 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
efivarfs on /sys/firmware/efi/efivars type efivarfs (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=22743)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
tracefs on /sys/kernel/trace type tracefs (rw,nosuid,nodev,noexec,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
ramfs on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)
/var/lib/snapd/snaps/bare_5.snap on /snap/bare/5 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/core20_1623.snap on /snap/core20/1623 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/core20_1852.snap on /snap/core20/1852 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/core22_607.snap on /snap/core22/607 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/firefox_1969.snap on /snap/firefox/1969 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/firefox_2487.snap on /snap/firefox/2487 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/gnome-3-38-2004_119.snap on /snap/gnome-3-38-2004/119 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/gnome-3-38-2004_137.snap on /snap/gnome-3-38-2004/137 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/gnome-42-2204_68.snap on /snap/gnome-42-2204/68 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/gtk-common-themes_1535.snap on /snap/gtk-common-themes/1535 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/snap-store_599.snap on /snap/snap-store/599 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/snap-store_638.snap on /snap/snap-store/638 type squashfs (ro,nodev,relatime,errors=continue,x-gdu.hide,x-gvfs-hide)

```

The last commands I used was *cd*, *mkdir* and *rmdir*. These commands allow you to enter directories, make folders, and remove folders respectively.

```
user@Lenovo-ideapad-330S-15ARR:~$ cd Desktop
user@Lenovo-ideapad-330S-15ARR:~/Desktop$ mkdir folder
user@Lenovo-ideapad-330S-15ARR:~/Desktop$ rmdir folder
user@Lenovo-ideapad-330S-15ARR:~/Desktop$
```



Windows OS Scripts

Here is my script written in bat, I used notepad++ to edit my code and displayed it in cmd, my script consisted of a way to add two numbers together, a function to see which of two numbers were bigger, and a script to store three places in an array.

I start the script with `@echo off` to make the text readable when its displayed in cmd, otherwise it echoes the file path for every line of code executed; then the user is prompted with a choice of one to three to do their desired task, if the user enters an invalid input it calls the menu again.

```

1  @Echo off
2
3  ::menu created with the goto commands that always points back to the menu.
4  ::menu selection done through user input and if statements
5  :menu
6  Echo Welcome to the menu!
7  Echo - - -
8  Echo Enter 1 to add or subtract two numbers!
9  Echo Enter 2 to check what two numbers are bigger!
10 Echo Enter 3 to store 3 places you've visited!
11 Echo - - -
12
13 Set /p select=What would you like to do?
14
15 if %select%==1 (goto numbers)
16 if %select%==2 (goto lessthan)
17 if %select%==3 (goto places)
18 Echo - - -
19 Echo This is not an option please try again!
20 Echo - - -
21 goto menu
22

```

```

Welcome to the menu!
- - -
Enter 1 to add or subtract two numbers!
Enter 2 to check what two numbers are bigger!
Enter 3 to store 3 places you've visited!
- - -
What would you like to do? 4
- - -
This is not an option please try again!
- - -
Welcome to the menu!
- - -
Enter 1 to add or subtract two numbers!
Enter 2 to check what two numbers are bigger!
Enter 3 to store 3 places you've visited!
- - -
What would you like to do? 1
- - -

```

If the user enters “1” into the menu it will use the *goto* command to take the user to the part of the script that adds two numbers together, this is done using variables by storing the users input, then using an *if* statement to either go to the add or subtract script. This also has a failsafe if the user doesn't pick a valid option, it will call *:numbers* again.

```

23  ::script used to add two numbers together using if statements
24  ::to point the script to the correct operator, + or -
25  :numbers
26  Echo - - -
27  Echo Please enter 2 numbers!
28  Set /p numb1=What is number 1?
29  Set /p numb2=What is number 2?
30  Set /p choice=Would you like to + or -?
31  Echo - - -
32
33  if %choice%==+ (goto add)
34  if %choice%==- (goto sub)
35  Echo This is not an option please try again!
36  goto numbers
37
38  ::+ script
39  :add
40  Set /a sum = %numb1% + %numb2%
41  Echo Your total is %sum%!
42  Echo - - -
43  goto menu
44
45  ::- script
46  :sub
47  Set /a sum = %numb1% - %numb2%
48  Echo Your total is %sum%!
49  Echo - - -
50  goto menu
51

```

```

- - -
What would you like to do? 1
- - -
Please enter 2 numbers!
What is number 1? 2
What is number 2? 4
Would you like to + or -? +
- - -
Your total is 6!
- - -

```

```

- - -
What would you like to do? 1
- - -
Please enter 2 numbers!
What is number 1? 7
What is number 2? 2
Would you like to + or -? -
- - -
Your total is 5!
- - -

```

If the user enters “2” into the menu it will take them to the part of the script that allows them to check two numbers against each other and output which one is bigger, this is done by taking the users input then calling a function that uses the *LSS* operator which is a simple Boolean that returns true or false when comparing two variables.

```

52 ::number test to check which is bigger using a function
53 :lessthan
54 Echo - - -
55 Echo Please enter 2 numbers!
56 Set /p numb1=What is number 1?
57 Set /p numb2=What is number 2?
58 Echo - - -
59 Call :function_lessthan numb1, numb2
60
61 ::the function for the number checker
62 :function_lessthan
63 if %numb1% LSS %numb2% (Echo %numb2% is bigger!) else (Echo %numb1% is bigger!)
64 Echo - - -
65 goto menu
66

```

```

- - -
What would you like to do? 2
- - -
Please enter 2 numbers!
What is number 1? 783
What is number 2? 2
- - -
783 is bigger!
- - -

```

If the user enters “3” into the menu it will take them to the final part of the script, this allows the user to enter three places they’ve visited and when, and then it puts it into a small sentence and stores it into an array. This is done by taking the initial input from the user then putting it into the array, then splitting the array structure into *place* and *time* attributes, this is useful as index 0 of both attributes are related to one another. Lastly the array is displayed to the user using a *for* loop which takes from the array attributes at index 0 and concatenates them into a sentence.

```

67 ::script to take user input for places they've been then store it in an array
68 ::and echo it back to them with a for loop
69 :places
70 Echo - - -
71 Set /p place1=Please enter the 1st place you've been!
72 Set /p time1=When did you go to %place1%?
73 Set /p place2=Please enter the 2nd place youve been!
74 Set /p time2=When did you go to %place2%?
75 Set /p place3=Please enter the 3rd place you've been!
76 Set /p time3=When did you go to %place3%?
77
78 Set array[0].place=%place1%
79 Set array[0].time=%time1%
80 Set array[1].place=%place2%
81 Set array[1].time=%time2%
82 Set array[2].place=%place3%
83 Set array[2].time=%time3%
84
85 Echo - - -
86 For /L %%i In (0 1 2) Do (
87     call Echo You went to %%array[%%i].place%% in %%array[%%i].time%%
88 )
89 Echo - - -
90 goto menu
91
92 Pause
93 Exit /B 0

```

```

- - -
What would you like to do? 3
- - -
Please enter the 1st place you've been! Canada
When did you go to Canada? January
Please enter the 2nd place youve been! Tokyo
When did you go to Tokyo? 2019
Please enter the 3rd place you've been! Tesco
When did you go to Tesco? the 2nd week of March
- - -
You went to Canada in January
You went to Tokyo in 2019
You went to Tesco in the 2nd week of March
- - -

```

Linux OS Scripts

Here is my script written in bash, I used an online code writing website called replit to write it which can be found at <https://replit.com/languages/bash>. My script includes a login page, a script to add a new user, a script to add multiple numbers together and a function to logout.

At the start of the script I declare an array with all the current users in it as strings, this is better than having individual variables as an array can be any length you want it to be, and it is easier to check if the username exists when it's entered which is shown in the user function. The *menu* function uses a case statement to give the user multiple options which then call the relevant function. It also validates the user's input if they select an unavailable option. This is all displayed to the user through the *echo* command

```

1  #declare array of users who can log in
2  declare userArray=("Danny" "James" "Gareth")
3
4  #menu function that gives the user options in
5  #a case statement which points to other functions
6  menu() {
7      echo "Please pick an option!"
8      echo " - - - "
9      echo "Enter 1 to create a new user!"
10     echo "Enter 2 to add some numbers!"
11     echo "Enter 3 to logout! "
12     echo " - - - "
13     read -p "What would you like to do? " choice
14     case $choice in
15         1)
16             user
17             ;;
18         2)
19             numb
20             ;;
21         3)
22             login
23             ;;
24         *)
25             echo " - - - "
26             echo "This is not an option!"
27             echo "Please try again !"
28             echo " - - - "
29             menu
30             ;;
31     esac
32 }
33

```

```

- - -
Please pick an option!
- - -
Enter 1 to create a new user!
Enter 2 to add some numbers!
Enter 3 to logout!
- - -
What would you like to do? 4
- - -
This is not an option!
Please try again !
- - -
Please pick an option!
- - -
Enter 1 to create a new user!
Enter 2 to add some numbers!
Enter 3 to logout!
- - -
What would you like to do? 1
- - -

```

If the user enters "1" in the menu it will call the *user* function. It will first echo the array of all current users, then allow the user to input the username of the new user. The script will then use an *if* statement to determine if the inputted name already exists, if it does exist it will run the *user* function again; if it does not exist then it will add the inputted data to the array and take the user back to the menu, this then allows the newly entered username to login to the program.

```
- - -
What would you like to do? 1
- - -
These are the current users:
Danny James Gareth
Please enter the new username! Gareth
- - -
This user already exists!
- - -
These are the current users:
Danny James Gareth
Please enter the new username! Charlie
New user added!
- - -
```

```
34 #function used to create new users
35 user() {
36     echo " - - - "
37     echo "These are the current users: "
38     echo "${userArray[*]}"
39     read -p "Please enter the new username! " newUser
40     if [[ ${userArray[@]} =~ ${newUser} ]]; then
41         echo " - - - "
42         echo "This user already exists!"
43         user
44     else userArray+=($newUser)
45         echo "New user added!"
46         echo " - - - "
47     fi
48     menu
49 }
50
```

If the user enters "2" in the menu it will call the *numb* function, the function starts by taking two inputs from the user being the *numb1* and *numb2* variables, I then use the *+* operator to add them both together and store it in another variable called *sum*. After echoing *sum* to the user the script will start an *until* loop which will run until the user enters "stop"; as you can see in the code, I've re used the variables *numb1* and *sum*, this is good coding practice as there was no need to create new variables as the other ones

```

51 #function used to add numbers together
52 numb( ) {
53     echo " - - - "
54     read -p "Please enter your first number! " numb1
55     read -p "Please enter your second number! " numb2
56     sum=$(( $numb1 + $numb2 ))
57     echo "Your current total is:" $sum
58     until [ "$numb1" = "stop" ]
59     do
60         read -p "Please enter another number, or stop! " numb1
61         sum=$(( $sum + $numb1 ))
62     done
63     echo " - - - "
64     echo "Your total is:" $sum
65     echo " - - - "
66     menu
67 }
68

```

```

- - -
What would you like to do? 2
- - -
Please enter your first number! 5
Please enter your second number! 4
Your current total is: 9
Please enter another number, or stop! 2
Please enter another number, or stop! 6
Please enter another number, or stop! stop
- - -
Your total is: 17
- - -

```

If the user enters "3" in the menu it will call the *login* function, this is also the function that is ran when the script first runs as seen on line 87 which is the bottom of the script. This script works by taking the users input and comparing it against the user array seen at the start of the script on line 2. If the username doesn't exist it will throw an error and call the *login* function again, if it matches then it calls the *menu* function. Below the code image you can see that adding a new user also works on the login page as the *user* function appends it to the array which *login* checks.

```

69  #first function thats called, allows an user from the
70  #declared array to log in and access the menu
71  login() {
72  echo " - - - "
73  echo "Welcome to my program!"
74  read -p "Please enter your username! " userInput
75  if [[ ${userArray[@]} =~ ${userInput} ]]; then
76  echo "Hello $userInput!"
77  echo " - - - "
78  menu
79  else
80  echo " - - - "
81  echo "This username is invalid, please try again"
82  login
83  fi
84  }
85
86  #call the login function to start the program
87  login

```

```

- - -
Welcome to my program!
Please enter your username! Gaynor
- - -
This username is invalid, please try again
- - -
Welcome to my program!
Please enter your username! Danny
Hello Danny!
- - -
Please pick an option!
- - -
Enter 1 to create a new user!
Enter 2 to add some numbers!
Enter 3 to logout!
- - -
What would you like to do? 1
- - -

```

```

- - -
These are the current users:
Danny James Gareth
Please enter the new username! Charlie
New user added!
- - -
Please pick an option!
- - -
Enter 1 to create a new user!
Enter 2 to add some numbers!
Enter 3 to logout!
- - -
What would you like to do? 3
- - -
Welcome to my program!
Please enter your username! Charlie
Hello Charlie!
- - -

```

Appendices

Bash Script

```
#declare array of users who can log in
declare userArray=("Danny" "James" "Gareth")

#menu function that gives the user options in
#a case statement which points to other functions
menu() {
    echo "Please pick an option!"
    echo " - - - "
    echo "Enter 1 to create a new user!"
    echo "Enter 2 to add some numbers!"
    echo "Enter 3 to logout! "
    echo " - - - "
    read -p "What would you like to do? " choice
    case $choice in
        1)
            user
            ;;
        2)
            numb
            ;;
        3)
            login
            ;;
        *)
            echo " - - - "
            echo "This is not an option!"
            echo "Please try again !"
            echo " - - - "
            menu
            ;;
    esac
}

#function used to create new users
user() {
    echo " - - - "
    echo "These are the current users: "
    echo "${userArray[*]}"
    read -p "Please enter the new username! " newUser
    if [[ ${userArray[@]} =~ ${newUser} ]]; then
        echo " - - - "
        echo "This user already exists!"
        user
    else userArray+=($newUser)
        echo "New user added!"
        echo " - - - "
    fi
}
```

```

    fi
    menu
}

```

```

#function used to add numbers together
numb() {
    echo " - - - "
    read -p "Please enter your first number! " numb1
    read -p "Please enter your second number! " numb2
    sum=$((numb1 + numb2))
    echo "Your current total is:" $sum
    until [ "$numb1" = "stop" ]
    do
        read -p "Please enter another number, or stop! " numb1
        sum=$((sum + numb1))
    done
    echo " - - - "
    echo "Your total is:" $sum
    echo " - - - "
    menu
}

```

```

#first function thats called, allows an user from the
#declared array to log in and access the menu
login() {
    echo " - - - "
    echo "Welcome to my program!"
    read -p "Please enter your username! " userInput
    if [[ ${userArray[@]} =~ ${userInput} ]]; then
        echo "Hello $userInput!"
        echo " - - - "
        menu
    else
        echo " - - - "
        echo "This username is invalid, please try again"
    fi
}

```

```

#call the login function to start the program
login

```

Bat Script

@Echo off

::menu created with the goto commands that always points back to the menu.

::menu selection done through user input and if statements

:menu

Echo Welcome to the menu!

Echo - - -

Echo Enter 1 to add or subtract two numbers!

Echo Enter 2 to check what two numbers are bigger!

Echo Enter 3 to store 3 places you've visited!

Echo - - -

Set /p select=What would you like to do?

if %select%==1 (goto numbers)

if %select%==2 (goto lessthan)

if %select%==3 (goto places)

Echo - - -

Echo This is not an option please try again!

Echo - - -

goto menu

::script used to add two numbers together using if statements

::to point the script to the correct operator, + or -

:numbers

Echo - - -

Echo Please enter 2 numbers!

Set /p numb1=What is number 1?

Set /p numb2=What is number 2?

Set /p choice=Would you like to + or -?

Echo - - -

if %choice%==+ (goto add)

if %choice%== - (goto sub)

Echo This is not an option please try again!

goto numbers

::+ script

:add

Set /a sum = %numb1% + %numb2%

Echo Your total is %sum%!

Echo - - -

goto menu

::- script

:sub

Set /a sum = %numb1% - %numb2%

Echo Your total is %sum%!

Echo - - -

goto menu

::number test to check which is bigger using a function

:lessthan

Echo ---

Echo Please enter 2 numbers!

Set /p numb1=What is number 1?

Set /p numb2=What is number 2?

Echo ---

Call :function_lessthan numb1, numb2

::the function for the number checker

:function_lessthan

if %numb1% LSS %numb2% (Echo %numb2% is bigger!) else (Echo %numb1% is bigger!)

Echo ---

goto menu

::script to take user input for places they've been then store it in an array

::and echo it back to them with a for loop

:places

Echo ---

Set /p place1=Please enter the 1st place you've been!

Set /p time1=When did you go to %place1%?

Set /p place2=Please enter the 2nd place youve been!

Set /p time2=When did you go to %place2%?

Set /p place3=Please enter the 3rd place you've been!

Set /p time3=When did you go to %place3%?

Set array[0].place=%place1%

Set array[0].time=%time1%

Set array[1].place=%place2%

Set array[1].time=%time2%

Set array[2].place=%place3%

Set array[2].time=%time3%

Echo ---

For /L %%i In (0 1 2) Do (

call Echo You went to %%array[%%i].place%% in %%array[%%i].time%%

)

Echo ---

goto menu

Pause

Exit /B 0

References

- [1] P. Pedamkar, "Functions of Operating System | 11 Best Functions of Operating System," 2020. Available: <https://www.educba.com/functions-of-operating-system/>.
- [2] Anonymous "Functions of Operating System," 2018. Available: <https://www.geeksforgeeks.org/functions-of-operating-system/>.
- [3] Anonymous (-04-06). *IBM Documentation*. Available: <https://www.ibm.com/docs/en/z-netview/6.2.1?topic=services-managing-inputoutput> [Accessed Apr 5, 2023].
- [4] Anonymous (-01-14T20:31:47+01:00). *Bash Functions*. Available: [/post/bash-functions/](#) [Accessed Apr 2, 2023].
- [5] Anonymous "Bash Scripting - Introduction to Bash and Bash Scripting," 2022. Available: <https://www.geeksforgeeks.org/bash-scripting-introduction-to-bash-and-bash-scripting/>.
- [6] C. Neagu, "20 best Command Prompt (CMD) commands you should know," 2022. Available: <https://www.digitalcitizen.life/best-cmd-commands/>.
- [7] M. Dancuk. (-01-11T15:26:10+00:00). *Windows CMD Commands: Mastering the Command Prompt*. Available: <https://phoenixnap.com/kb/cmd-commands> [Accessed Apr 2, 2023].