Command Line Interfaces

"A command-line interface, or CLI, is a method for users to issue text based commands to a computer system"

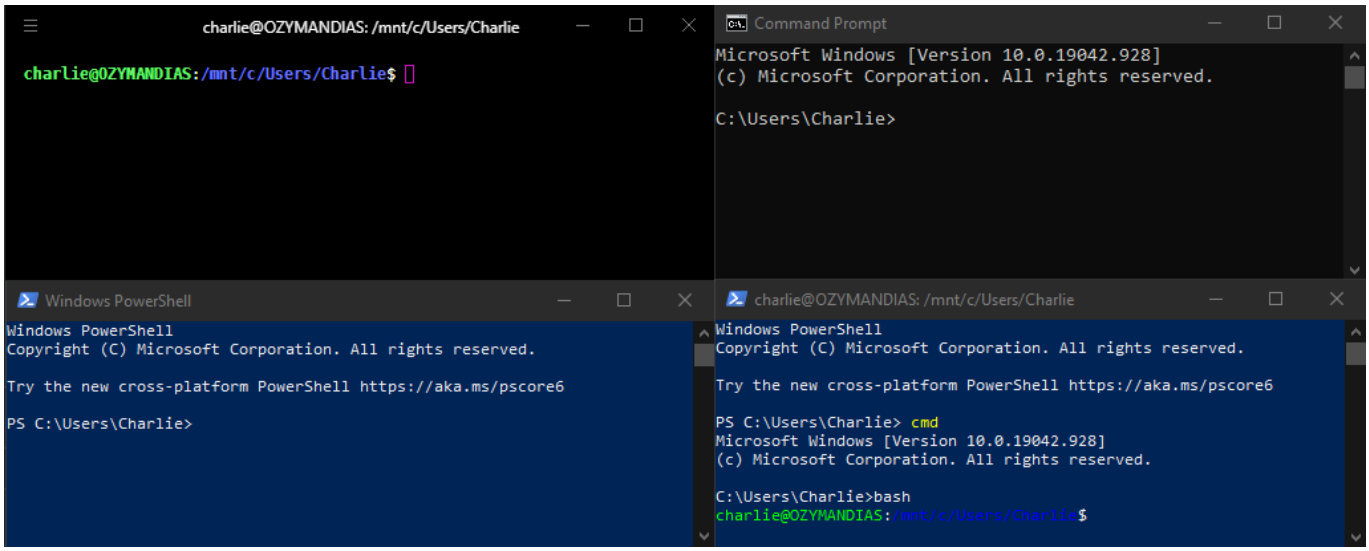
Most users of modern computer systems interact with software using a graphical user interface, or GUI. Before the widspread use of GUIs, computers were typically utilised by issuing lines of text to the computer as command-line operations, which were then processed by the computer. Although many programs can provide their own command line interface, many leverage the CLI provided by the operating system on their device. Other common names for command line interface may include terminal, terminal emulator, console, shell, command line interpreter, or interactive shell.

Common shortcomings of the command lines usability include remembering the available commands and how to use them, as well as their requirements for syntax adherance, which means that typing errors result in program errors. Some scenarios where CLIs are very useful include:

- Replication: It is easy to record and recall a series of commands in logical sequence
- Configuration: It is trivial to duplicate lines of commands and swap out single parameters or minor differences
- Flexibility: It often allows many additional parameters to be passed to a program at startup, such as starting a web browser in private browsing mode
- Automation: Command line programs can automate scheduled tasks
- GUI availability: A program may often be released before anyone has taken the time to build a GUI for it

Most operating systems have their own form of commandline console, such as:

- Windows (Legacy): cmd, or command prompt
- Windows (Modern/ Cross-Platform): Powershell
- Linux (debian): Bash (bourne again shell)
- MacOS: Bash/ Z shell



You try:

Goal: To explore how to issue commands to a computer

Open your operating systems command line interface or terminal. If you are unsure how to do this, try searching the internet for instructions on your operating system.

- Try issuing your first instruction. A good one to start with is `help`
- You might notice that the CLI doesn't incrementally adjust for the flood of information returned, resulting in a lot of information overflow. You can try step through each new line with the inclusion of an additional *piped* command appended to the end, like `help | more`
- Most programs accessible from the command line include some useful help information which can be accessed by appending the *help* argument. In linux this is often seen as the flag `--help` and on windows the `/?` is often used.
- Try use the dir command to list directory contents. You can access the help using `dir --help` or `dir /?` depending on your operating system

The commands available and the syntax (the structure and formatting of how instructions are written) will vary based on the system being used. Many systems offer multiple ways of performing the same commands, often termed *aliasing*, which simplifies the command issuing process and prevents errors. For example, the following examples illustrate the same instructions being issued on different CLIs.

Command-Prompt:

```
echo Some Text

set VARONE=Variable One Content

echo %VARONE%
```

Bash:

```
echo Some Text

VARONE=Variable One Content

echo $VARONE
```

Powershell:

```
Write-Output('Some Text')

echo $VARONE

$VARONE = "Variable One Content"
```

```
Set-Variable -Name "VARTWO" -Value "Variable Two Content"

Write-Output($VARTWO)
```

More about

Most instructions or instruction sequences contain the following components:

- Commands: A specific command that is understood by the interpreter, such as changing directory or creating a file
- Environment variables: Global or local variables which determine the current context of operation, such as the path to the users home directory
- Programmes
- Arguments, named parameters or 'flags'

One concept imperative to proper use of the command line is undertanding file paths and directory structures.

In the command line, often the "relative" path, or the file path of the "current" directory, is represented as a dot or fullstop, `.`
Using a double dot, `..`, indicates the parent directory.

You can try see how this works by using the command `cd ..` to move to the parent directory of the current path

Paths are structured a bit differently between different operating systems.

On Windows, the system directory is usually *C:\Windows* and the home directory of the current user is *C:\Users\Username*

The windows commandline displays the currently active path in the follwoing format `C:\File\Path>` whilst Windows PowerShell differentiates itself from the CMD interface by prefixing the path with PS such as in `PS C:\File\Path>`.

Linux typically identifies the user, host and path in a format such as `currentuser@hostname:/file/path/$`. Linux also typically uses the path `/` for the system directory and `/home/username/`, usually represented by the tilde character `~/` for the currently active user directory.

Linux does not mount separate disks like windows, and to change disks in windows CMD users will have to input the drive letter followed by a colon, such as `d:` in order to traverse the filesystem on that drive.

Instructions are commonly used in a read–eval–print loop (REPL) format. This process executes in the way it is described, in that it reads the inputs, evaluates the response and returns it (print or echo).

Command-Prompt:

```
set "PREFIX=Well, hello there"

set "SUFFIX=!"

set /p NAMEVAR="What is your name: "

echo %PREFIX% %NAMEVAR%%SUFFIX%
```

Bash:

```
PREFIX="Well, hello there"

SUFFIX="!"

read -p "What is your name: " NAMEVAR

echo $PREFIX $NAMEVAR$SUFFIX
```

Sometimes commands on different systems may require the use of different flags. The following example sends an ICMP request to a server and waits for the response. On Windows, you specify the **N**umber of times to ping the server, whereas on Linux you specify the **C**ount of pings. The concepts are usually similar, however the syntax may vary. You can use the `--help` or `/?` commands to learn how to use the command on each system.

Command-Prompt:

```
ping google.com -n 10
```

Bash:

```
ping google.com -c 10
```

You can also cancel currently executing commands, such as using the CTRL+C keyboard shortcut.

Commandline instructions can also be written into system executable's. These are simply text files with instructions that tell the interpreter what instructions to execute and in which order.

On Windows, these files typically carry file extensions like .bat .cmd and are often called "batch" files.

On Linux, the convention is to use .sh as a file extension but any text file can be marked as an executable.

You can then use programs like cron or the windows task scheduler to run these instructions on a schedule or at predetermined times.

Similar instructions can also be bundled into a configuration file and executed by another program, such as a makefile or python script.

Check your knowledge:

1. A CLI is:
    a. *A proprietary program released by Microsoft for the Windows platform*
    b. *A way to interact with a computer system or program using text based commands*
    c. *The console or terminal on an operating system*
2. Using a CLI allows users to:
    a. *Write their own programs and execute any instruction*
    b. *Browse the internet without leaving data on the host system*
    c. *Use the available programs and commands in a structured manner*
3. CLI variables are:
    a. *Identified, declared, and accessed differently depending on the system used*
    b. *Always prefixed with a $ symbol*
    c. *Accessed using the %VARIABLENAME% syntax*
    d. *Template data structures which can be referenced with {{ VARIABLENAME }} syntax*

Module video tutorial:

- https://youtu.be/NrINzV8UUi8

Further reading:

- CMD Cheatsheet: https://dev.to/moniruzzamansaikat/windows-command-cheatsheet-hce
- Bash Cheatsheet: https://www.codecademy.com/learn/learn-the-command-line/modules/learn-the-command-line-navigation/cheatsheet
- Setting the windows PATH: https://www.computerhope.com/issues/ch000549.htm