# Introduction to Computing

The Terminal

Cain Susko

2023-06-16

## Commands and The Command Line

The Terminal is known by many names – including (but not limited to) a *shell*, *command prompt*, or *command line*. It is the most powerful way to interact with your computer – however it takes time to learn how to use it effectively.

Note: When I say Command Line, or Terminal etc. I mean `POSIX` compliant terminals. `POSIX` is a standard which ensures that operating systems can easily communicate with each other. Every common Operating System is `POSIX` compliant ACCEPT WINDOWS `:(`. Thus, the following information is generally *not* applicable to standard windows computers. Luckily, there are many ways to use a `POSIX` compliant terminal on Windows, either by installing Windows Subsystem for Linux (WSL), or using websites like https://bellard.org/jslinux/

Generally, a command – which you type into a command line – is structured like so:

```
someCommand --option input
```

Where `someCommand` is the program you are *calling*, the `--option` is the way you would like to use the program, and `input` is what you want the program to do something to.

### Programs

A command, or *program*, is a piece of code on your computer which has been made to do something. These programs can be in any language as long as they have been *compiled* – or formatted – for your operating system .

Your computer knows which programs you can run by looking at it's `PATH`, which is a variable on your computer that indicates where in your *file system* programs are stored. Normally, your path should already be set to something by your operating system, but for example, if my `PATH` was:

```
PATH = /home/cain/programs
```

And I had a program in `/home/cain/programs` (which is a directory) called `coolProgram` – then if I wrote on the command line

```
coolProgram
```

It would run the program!

## The File System and Paths

As we just saw, your computer has a `PATH` variable – but what is this variable representing? In fact, it represents *file paths* within your *file system.* The file system begins at the *root*, which is a directory denoted by just `/`. Within the root there are many other directories, like `home` – who's path is `/home`, and contains the files of each user on the computer. For example, all my files that I create are within

```
/home/cain/

NOTE: /home/user/ is normally denoted by: ~
This is called the Home Directory
```

And within that directory, there are many others! But directories can also store files, which are what you're probably looking for anyways.

To see the directory you're currently in, use the command:

```
pwd
```

Short for *Print Working Directory.*
To see the contents of your working directory, such as files or other directories, use the command

```
ls
```

Short for *LiSt*
And to go into a different directory, use the command:

```
cd aDirectory
```

Short for *Change Directory.*

Both `cd` and `ls` can take a *file path* as input in order to use the command on a specific directory which may not be in your *working directory.* These file paths can either be **absolute**, meaning they start with the *root*, or **relative**, meaning they start from your *working directory.* For example, if we want to list the contents of the directory `/home/cain/documents`, and our working directory is `/home/cain` (aka `~`, or the Home Directory) we could either use:

- Absolute: `ls /home/cain/documents` (or `ls ~/documents`)
- Relative: `ls documents`

Furthermore, when using **relative** paths, your current directory is denoted by `./`. Additionally, the "parent" directory is denoted by `../`. For example, if my *working directory* is `~` (aka `/home/cain`) then:

- `cd ./` does not change my working directory
- `cd ../` changes my working directory to `/home`

These concepts and commands should give you the tools you need to navigate and view your file system from within the terminal.

## Creating and Deleting Files

But what about making new files or editing existing ones? This too can be accomplished on the terminal!

In order to create a new file, use the command

```
touch path/to/newFileName.type
```

This will create a new file at the specified path (Note that `newFile.type` is a valid *relative* path).
To create a new directory, use the command

```
mkdir path/to/newDirectoryName
```

To then delete a directory, use the command

```
rmdir path/to/directoryName
```

And to delete a file, use the command

```
rm path/to/file.type
```

## Editing Files

Now that we can navigate our file system, create, and delete files – how do we add things to them and change them? In fact, we can do this on the terminal too!

In order to edit files, we must use a **text editor**. You may be familiar with *word processors*, like Microsoft word, but these have a ton of stuff going on in the background to format the text and make it look pretty. A text editor instead only shows the text, with minimal formatting – like changing the colour of text to make programming easier.

There are many different text editors one can choose from, but there is usually always one pre-installed by your operating system. On Linux computers, `nano` and `vim` are the most common. `nano` is easy to use without much experience, whereas `vim` requires a lot more practise to use effectively.

To open a file with a text editor, like `nano`, use the following command:

```
nano path/to/file.type
```

`nano` then displays the contents of the file, and you can edit it and save it as needed.

## Exercises

These exercises are meant to be done in order.

### Easy

0. Determine what your working directory is.
1. List the contents of your working directory
2. Go to the "root" of the file system, and then go back to your Home Directory.
3. In your Home Directory, make a new directory called `programs`.
4. change directories into the newly created `programs` directory.
5. Create a file in `programs` called `myScript.sh`

### Medium

0. Open `myScript.sh` using `nano`.
1. type `echo "Hello World"` into the file.
2. Save the changes you made, and close `nano`,
3. use the command `chmod +x myScript.sh` to make your new program callable
4. Run your program by *calling* it, using the command `myScript.sh`

### Hard

0. Google how to use variables in a `Bash Script`
1. Create a script that can take an input, such as `cain` and print `Hello cain`.

Note: For these exercises, it is best to use https://bellard.org/jslinux/ and specifically the second to last option in the list (Fedora 33 Linux, Console).

If you have a Linux or Mac computer, or you have WSL installed, the terminals already on these systems should work great too!