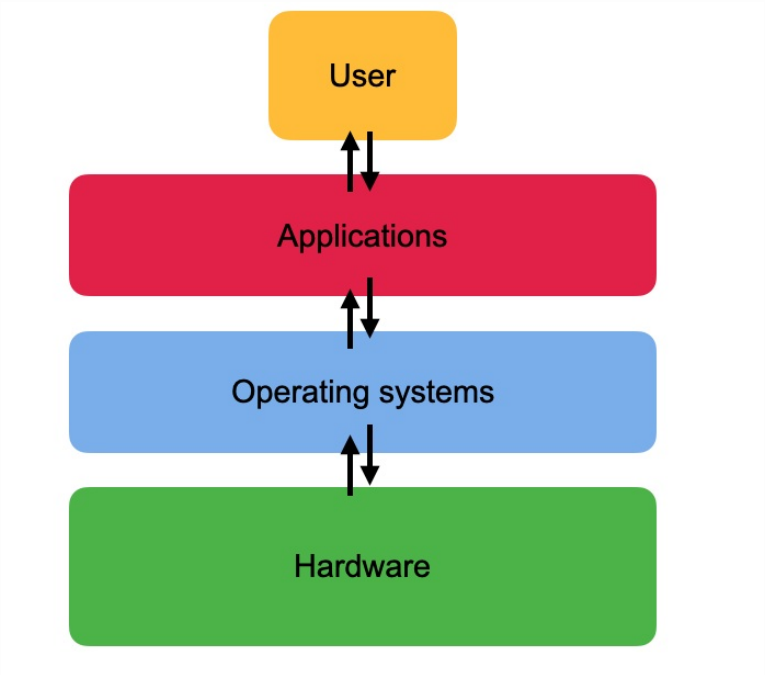
Learning Objectives

Learners will be able to...
1. Identify what tasks/functions an operating system does.
1. Describe the history of Unix systems
1. Use the appropriate commands on the terminal to change directories, list the contents of a directory, and list the contents of a file.

What is an Operating System?

The **operating system** (OS) is the software that sits between the computer hardware and the applications running on a computer.



The User interacts with applications which interact with the operating system which interacts with hardware

Most modern operating systems (e.g. MacOS, iOS, Android, Microsoft Windows) provide a graphical user interface, but some only provide a text-based interface.

Try out interacting with the text-based interface on the left by typing `echo "Welcome to Unix"` and pressing the Enter or Return key.

- You should see **Welcome to Unix** print out below the command you typed.
- You can try printing out other phrases by replacing the text in the double quotes like so: `echo "Are you ready?"`.

Operating systems allow users to multi-task - for example, you can watch a video and receive texts at the same time - by carefully managing a wide range of system resources.

What does an operating system do?

- Manage memory allocation for running applications
- Process scheduling; managing CPU (central processing unit) allocation for running processes
- Manage data storage
- File management
- Manage Input/Output (I/O), specifically process input from devices such as keyboards and touchpads, send output to a display
- Manage peripherals such as printers and scanners

Unix is a **multi-user** operating system. Under Unix, each user can access and share system resources such as memory. In contrast, other operating systems, like Windows, start a new session with distinct resources for each user.

The UNIX operating system

The first implementation of UNIX was developed at Bell Labs, a division of AT&T, in 1969. It was first developed for the PDP-7, a mini-computer from Digital Equipment Corporation (DEC). It was then rewritten in assembly language for the DEC PDP-11.

The C programming language, one of the first high-level programming languages, was also developed at Bell Labs and in 1973 much of the UNIX kernel was rewritten in C. The fact that UNIX was written in a high-level language, rather than assembly language, meant it could be ported to other hardware systems without a complete rewrite. This is easy to imagine now - consider how a single app on the App Store (for iPhones/iPads) or the Play Store (for Android phones and tablets) can work across different phones and tablets.

Also, because C was written with the idea of using it for implementing the UNIX kernel, it was designed to be small and efficient, ideal features for an operating system.

Expanding out

AT&T was deemed by the US government to be a monopoly, and one consequence was that they were not allowed to sell the UNIX operating system. Since they could not sell UNIX, AT&T leased the OS, including the source code, to universities. With access to the code, computer science students could learn about the operating system and experiment with modifying it. Many of these students carried this knowledge with them into the professional world and contributed to the expanding popularity of UNIX.

The University of California at Berkeley was a big player in the development of UNIX in the late ’70s and through the mid ’80s. The Berkeley Software Distribution (BSD) variant of UNIX contributed many new features including an improved file system, virtual memory management, a complete TCP/IP implementation and a variety of networking tools.

With the breakup of their monopoly in the early ’80s, AT&T was free to market their version of UNIX to commercial vendors. They employed developers to enhance UNIX and also incorporated some of the BSD features. AT&T released System V of Unix in 1989 and it formed the basis for many commercial implementations of UNIX such as Solaris for Sun workstations and Ultrix for Digital computers.

Standardization

With some UNIX implementations based on BSD and others based on System V and the variability of added features provided by computer vendors, UNIX needed standards so that applications could be shared across platforms. The first step in standardizing UNIX began with the movement towards standardizing the C language in 1989. The standardization guaranteed that a C program that just used the C libraries could run on any computer, using any operating system.

There are two UNIX standards, "Single UNIX Specification (SUS)" and "POSIX". Those that pass conformance tests for the SUS are granted the right to be branded "UNIX" by the holders of the UNIX trademark. The POSIX standard was developed by the Institute of Electrical and Electronics Engineers (IEEE). Apple's macOS follows the POSIX standard.

Linux

Linux is an open-source version of the UNIX operating system. **Open source** means it is developed by a community and anyone can view or modify the source code. The open source nature of this operating system has led to many different versions of it. These versions are called **distributions**, often abbreviated as distro. There are different distributions of Linux tailored for different hardware systems from the Raspberry Pi to versions that run on supercomputers. Linux is considered mostly POSIX compliant but is not POSIX certified.

There are dozens of Linux distributions, but some of the more popular ones you will hear are Debian, Red Hat, Fedora, and Ubuntu. Different distributions may differ in which applications are included, whether they are completely open source, and how often a stable version is released.

The distribution of Linux you will access through the terminal in this course is Ubuntu 18.04.6 (which is based on the Debian distribution of Linux). You can see this version at the very top of the **message of the day** (the message printed when you start the terminal):

```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1058-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

For the rest of this course, we will automatically `clear` the message of the day so you have more screen space.

Try it out for yourself by typing `clear` into the terminal on the left and pressing Enter.

Linux Kernel, Tools and Libraries

The portion of the operating system that handles the computer resources (CPU, RAM and devices) is referred to as the **kernel**. The initial version of the Linux kernel, created by Linus Torvalds, was written specifically for the Intel 80386 chip, a standard chip used in personal computers in the late 1980's and 1990's. The kernel has been ported to many different chip architectures since then.

There is also a set of standard software utilities that accompany the system that provide services such as a command line interpreters, file utilities and editors that we will cover in this module. Many of the Linux tools are written in C and are therefore machine independent.

Tux, the Linux mascot

When people refer to the Linux operating system they mean the kernel as well as the accompanying set of tools and libraries that is actually called GNU. Some people refer to this combination as GNU/Linux (as you see in the message of the day) but we will just refer to the kernel plus the utilities as Linux. For the most part the utilities that are packaged with Linux are functionally equivalent to the utilities with the same name in UNIX, but they have been rewritten.

The Command Line Interface

The Command Line Interface, better known as CLI, or Shell is one way to interact with your computer. The command line interface which you see in the terminal window on the left is called **Bash**. It is the default login shell for most Linux distributions. It is considered very portable and is widely used on many UNIX implementations as well as on other operating systems.

Command Line Prompt

In the terminal window on the left you will see the **command line prompt**. It looks something like what you see below:

```
codio@matrixrudolf-trustgame:~/workspace$
```

Take a look at the different pieces of the command prompt:
- The first part of the command prompt is the **username** followed by the @ symbol. On the Ubuntu server you see in the terminal, Codio has automatically created a user called `codio`.
- The **server address** after the @ sign is specified using a unique combination of words. Codio runs each assignment/lesson as a distinct Ubuntu so you will notice that the server address changes throughout this course.
- The information after the colon is an abbreviation of the **current working directory**.
- The command line prompt ends with a `$`

Command prompt colored to highlight features as described in list above

info

Folders vs. directories

The terms **folders** and **directories** are sometimes used interchangeably.
In general the term **folders** is used in conjunction with graphical user interfaces and the term **directories** is more common with a CLI.

Home and Working Directories

The ~ symbol, better known as the *tilde symbol* represents the **home directory**. This is why the ~ directory is `/home/codio`.

Remember that Unix, and therefore Linux, is a **multi-user** operating system, meaning each user can access and share system resources such as memory. To keep things organized, each user has their own home directory. If you were on a server with multiple users, each one would have their own home directory as in:

```
/home/username1
/home/username2
```

Another important directory is the **working directory**. The working directory is basically where you are (the directory you are currently working in).

While you can see the relative working directory after the : in the command line prompt, sometimes it's helpful to **p**rint the **w**orking **d**irectory.

Type `pwd` (which stands for print working directory) in the terminal window to the left.

```
pwd
```

The shell will print the working directory directly below the command:

```
codio@matrixrudolf-trustgame:~/workspace$ pwd
/home/codio/workspace
```

Print working directory

Printing the contents of a file and viewing the manual

`cat`is the shell command - it is short for “catalog” and it prints out the contents of a file.

In the terminal window on the left type:

```
cat /etc/os-release
```

`/etc/os-release` is the file we are printing out. In a directory called `etc` there is information about the release we are using, in a file called `os-release`.

You should see something like this:

```
NAME="Ubuntu"
VERSION="18.04.6 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.6 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

The general syntax for `cat` is:

Viewing the `man`ual

You can find more information about many bash commands by typing `man` followed by the command name, `man` is short for **manual**.

Try it now by typing `man cat` in the terminal window on the left.

- To exit the manual, type the letter `q`.

Type `man bash` in the terminal window on the left and use the information provided to answer the question below.

- Scroll the manual page by using the up and down arrow keys or the scroll wheel on your mouse
- To exit the manual, type the letter `q`.

Listing Directory Contents

Try the following versions of the ls command, one at a time. Compare what you see in the output with what is visible in the file tree panel on the far left.

- `ls`
- `ls -l`
- `ls -a`
- `ls -m`
- `ls -R`

▼ **What do the characters at the beginning of the lines mean?**

When you get a full listing of the directory contents you see each line start with something like `-rwxr-xr-x`. This is a topic we will cover extensively in the **Processes, Services and Privileges** but in short, these are the privileges for the file. The `r` stands for **read**, the `w` stands for **write**, and the `x` stands for **execute**. The three sets are associated with the **owner**, the **group** and **everyone**. Where you see a `d` as the first letter, that means it is a directory.

Did you notice how the argument to the `ls` command changed the output?
Command option	Output
———	———-
(none)	A regular sorted list of files in the directory
-l	A more complete list with permissions
-a	Hidden files were included
-m	A comma-separated list
-R	Included directories and their contents

The `-R` argument stands for **recursive**. This is a very important argument when you are dealing with directories because you can have many nested directories and you may want your commands to apply to them all.

A closer look at the ls -a option

Type in `ls -a` again, you should see something similar to the image below.

directory listing using -a

▼ **Can you guess which are files and which are directories?**

The directories are the items listed in blue.

As you can see, *bash* sorts files and directories alphabetically starting with the dots.

info

Single and double dots

Dots in bash are useful utilities that help with navigating the file system using the CLI.

In an `ls -a` listing...

. refers to the current directory.
.. refers to the parent directory.

There are a lot of arguments you can pass to `ls`, to see them all type `man ls`.

Navigating the File System

We learned about the home directory when we discussed the command line prompt, there is another important directory to mention, the **root directory**.

Type in the command we learned earlier:

```
pwd
```

You will see the following:

```
/home/codio/workspace
```

The leading `/` means **root**.

Type the following command at the prompt:

```
ls /
```

This command shows you all the directories and files that are in the root directory.

info

Terminal Tips

1. Cycle through all the commands you have entered using the up and down arrow keys
2. Position your cursor at the beginning and end of the command prompt by using ctrl-a and ctrl-e
3. Use the tab key to complete directory and filenames

Try these variations

Use the tips above to reduce typing.
- `ls /home`
- `ls /home/codio`
- `ls /home/codio/workspace`

The last one is equivalent to just typing `ls` since `/home/codio/workspace` is the folder you are in.

Changing Directories

The command to change directories is `cd` as in “change directory”. To try it out we’ll switch to the directory `test`.
Type the following command at the prompt:

```
cd test
```

Now you can type in `ls` to see what is in that directory and then `cat` followed by the name of the file that is in there. You have now explored the test directory!

challenge

What happens if you type in the change directory command again?

```
cd test
```

▼ **Why is this an error?**

You get an error: `bash: cd: test: No such file or directory`.

We are already in the test directory and there is no test directory inside it.

A relative path is related to the current working directory. When we typed in `cd test` that was a relative path. We were switching to a directory that was inside of our current directory so we didn't have to list out the entire path starting at the root.

An **absolute path** is represented as starting from the root directory by leading with the `/`. The absolute path equivalent of the `cd test` we did earlier would be `cd /home/codio/workspace/test`

To back out of the `test` directory, you could use absolute file paths (`cd /home/codio/workspace`) or you can use the relative path. Earlier we discussed the single and double dot files you see when you type the command `ls -a`. The double dot file is very useful.

Go up one directory by typing:

```
cd ..
```

You should find yourself back in `/home/codio/workspace` (which you can verify with `pwd`).