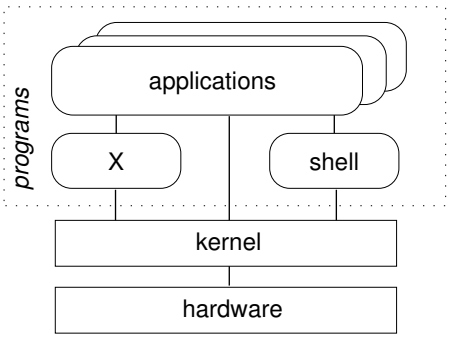
Module 1

Introduction

1.1 Unix and Linux

- Linux is based on Unix
  - Unix philosophy
  - Unix commands
  - Unix standards and conventions
- There is some variation between Unix operating systems
  - Especially regarding system administration
  - Often Linux-specific things in these areas

1.2 Unix System Architecture



- The shell and the window environment are programs
- Programs' only access to hardware is via the kernel

1.3 Unix Philosophy

- Multi-user
  - A **user** needs an **account** to use a computer
  - Each user must **log in**
  - Complete separation of different users' files and configuration settings
- Small components
  - Each component should perform a single task
  - Multiple components can be combined and chained together for more complex tasks
  - An individual component can be subsituted for another, without affecting other components

1.4 What is Linux?

- Linux kernel
  - Developed by Linus Torvalds
  - Strictly speaking, 'Linux' is just the kernel
- Associated utilities
  - Standard tools found on (nearly) all Linux systems
  - Many important parts come from the **GNU** project
    - Free Software Foundation's project to make a free Unix
    - Some claim the OS as a whole should be 'GNU/Linux'
- Linux distributions
  - Kernel plus utilities plus other tools, packaged up for end users
  - Generally with installation program
  - Distributors include: Red Hat, Debian, SuSE, Mandrake

1.5 Using a Linux System

- Login prompt displayed
  - When Linux first loads after booting the computer
  - After another user has logged out
- Need to enter a **username** and **password**
- The login prompt may be graphical or simple text
- If text, logging in will present a **shell**
- If graphical, logging in will present a **desktop**
  - Some combination of mousing and keystrokes will make a **terminal window** appear
  - A shell runs in the terminal window

1.6 Linux Command Line

- The shell is where commands are invoked
- A command is typed at a **shell prompt**
  - Prompt usually ends in a dollar sign (`$`)
- After typing a command press `Enter` to invoke it
  - The shell will try to obey the command
  - Another prompt will appear
- Example:

```
$ date
Thu Jun 14 12:28:05 BST 2001
$
```

  - The dollar represents the prompt in this course — do not type it

1.7 Logging Out

- To exit from the shell, use the `exit` command
- Pressing `Ctrl+D` at the shell prompt will also quit the shell
- Quitting all programs should log you out
  - If in a text-only single-shell environment, exiting the shell should be sufficient
  - In a window environment, the window manager should have a log out command for this purpose
- After logging out, a new login prompt should be displayed

1.8 Command Syntax

- Most commands take **parameters**
  - Some commands *require* them
  - Parameters are also known as **arguments**
  - For example, `echo` simply displays its arguments:

```
$ echo

$ echo Hello there
Hello there
```

- Commands are case-sensitive
  - Usually lower-case

```
$ echo whisper
whisper
$ ECHO SHOUT
bash: ECHO: command not found
```


1.9 Files

- Data can be stored in a **file**
- Each file has a **filename**
  - A label referring to a particular file
  - Permitted characters include letters, digits, hyphens (`-`), underscores (_), and dots (`.`)
  - Case-sensitive — *NewsCrew.mov* is a different file from *NewScrew.mov*
- The `ls` command lists the names of files

1.10 Creating Files with `cat`

- There are many ways of creating a file
- One of the simplest is with the `cat` command:

```
$ cat > shopping_list
cucumber
bread
yoghurts
fish fingers
```

- Note the greater-than sign (`>`) — this is necessary to create the file
- The text typed is written to a file with the specified name
- Press `Ctrl+D` after a line-break to denote the end of the file
  - The next shell prompt is displayed
- `ls` demonstrates the existence of the new file

1.11 Displaying Files' Contents with `cat`

- There are many ways of viewing the contents of a file
- One of the simplest is with the `cat` command:

```
$ cat shopping_list
cucumber
bread
yoghurts
fish fingers
```

- Note that no greater-than sign is used
- The text in the file is displayed immediately:
  - Starting on the line after the command
  - Before the next shell prompt

1.12 Deleting Files with `rm`

- To delete a file, use the `rm` ('remove') command
- Simply pass the name of the file to be deleted as an argument:

```
$ rm shopping_list
```

- The file and its contents are removed
  - There is no recycle bin
  - There is no 'unrm' command
- The `ls` command can be used to confirm the deletion

1.13 Unix Command Feedback

- Typically, succesful commands do not give any output
- Messages are displayed in the case of errors
- The `rm` command is typical
  - If it manages to delete the specified file, it does so silently
  - There is no 'File shopping_list has been removed' message
  - But if the command fails for whatever reason, a message is displayed
- The silence can be be off-putting for beginners
- It is standard behaviour, and doesn't take long to get used to

1.14 Copying and Renaming Files with `cp` and `mv`

- To copy the contents of a file into another file, use the `cp` command:

```
$ cp CV.pdf old-CV.pdf
```

- To rename a file use the `mv` ('move') command:

```
$ mv commitee_minutes.txt committee_minutes.txt
```

  - Similar to using `cp` then `rm`
- For both commands, the existing name is specified as the first argument and the new name as the second
  - If a file with the new name already exists, it is overwritten

1.15 Filename Completion

- The shell can making typing filenames easier
- Once an unambiguous prefix has been typed, pressing `Tab` will automatically 'type' the rest
- For example, after typing this:

  ```
  $ rm sho
  ```

  pressing `Tab` may turn it into this:

  ```
  $ rm shopping_list
  ```

- This also works with command names
  - For example, `da` may be completed to `date` if no other commands start 'da'

1.16 Command History

- Often it is desired to repeat a previously-executed command
- The shell keeps a **command history** for this purpose
  - Use the `Up` and `Down` cursor keys to scroll through the list of previous commands
  - Press `Enter` to execute the displayed command
- Commands can also be edited before being run
  - Particularly useful for fixing a typo in the previous command
  - The `Left` and `Right` cursor keys navigate across a command
  - Extra characters can be typed at any point
  - `Backspace` deletes characters to the left of the cursor
  - `Del` and `Ctrl+D` delete characters to the right
    - Take care not to log out by holding down `Ctrl+D` too long

1.17 Exercises

1. a. Log in.
   b. Log out.
   c. Log in again. Open a terminal window, to start a shell.
   d. Exit from the shell; the terminal window will close.
   e. Start another shell. Enter each of the following commands in turn.
      - `date`
      - `whoami`
      - `hostname`
      - `uname`

- `uptime`

2. **a.** Use the `ls` command to see if you have any files.

   b. Create a new file using the `cat` command as follows:

   ```
   $ cat > hello.txt
   Hello world!
   This is a text file.
   ```

   Press `Enter` at the end of the last line, then `Ctrl+D` to denote the end of the file.

   c. Use `ls` again to verify that the new file exists.

   d. Display the contents of the file.

   e. Display the file again, but use the cursor keys to execute the same command again without having to retype it.

3. **a.** Create a second file. Call it *secret-of-the-universe*, and put in whatever content you deem appropriate.

   b. Check its creation with `ls`.

   c. Display the contents of this file. Minimise the typing needed to do this:

   - Scroll back through the command history to the command you used to create the file.
   - Change that command to display *secret-of-the-universe* instead of creating it.

4. After each of the following steps, use `ls` and `cat` to verify what has happened.

   a. Copy *secret-of-the-universe* to a new file called *answer.txt*. Use `Tab` to avoid typing the existing file's name in full.

   b. Now copy *hello.txt* to *answer.txt*. What's happened now?

   c. Delete the original file, *hello.txt*.

   d. Rename *answer.txt* to *message*.

   e. Try asking `rm` to delete a file called *missing*. What happens?

   f. Try copying *secret-of-the-universe* again, but don't specify a filename to which to copy. What happens now?

