Module Title:	Operating Systems & Computer Networks
Module Code:	CST2555
Module Leader:	Dr Ian Mitchell

Bash Script
Coursework 1

Student Number:	.....
Tutor:	.....

Computer Science

2021-22

COVID-19

Every effort has been made to ensure that this coursework can be completed on the Virtual Machine (VM) provided. This means that in order to complete the coursework you do not require any proprietary software. All software is provided on the VM, this plan has been implemented due to the uncertainty and disruption that COVID-19 may bring during the delivery of this coursework.

Abstract

The aims of this coursework are for the student to demonstrate knowledge and skills in developing a solution satisfying the problem definitions.

Contents

Chapter 1

Introduction

The problems included here must be solved using *only* the following:

1. `vim` editor
2. `bash` shell script
3. `linux` commands

Failure to follow these will result in a loss of marks.

All the coursework can be completed from studying the lecture notes and accompanying exercises. Additional recommended readings are [2, 3, 6, 7].

This coursework may seem more complex than other courseworks you have completed or attempting. It is not, in fact it requires a very simple solution. The issue is trying to explain the problem, the module leader will provide demonstrations of the problem domain and how the bash script is expected to run in various lectures. Of course there is ample opportunity to ask questions about aspects of the coursework you don't quite understand in the formative feedback session arranged and in every lecture before the submission date.

The structure of this document is as follows:

Chapter 1 Introduction.

Chapter 2 Create Files problem specification.

Chapter 3 Read Files problem specificaiton.

Chapter 4 Evaluation instructions.

Chapter 5 Documentation instructions.

Chapter 6 TarBall and compression instructions.

Chapter 7 Formative and Summative Feedback and Assessment.

Chapter 8 Assessment Criteria and Rubric.

1.1 Marks and Assessment weighting

This coursework is worth 20% of the module and is marked out of 30.

The assessment criteria is in table 8.

1.2 Deadlines and Submission

Read the following carefully.

- The coursework deadline is by **23:59hrs (local time)** on **09/01/2022**.
- The coursework is to be submitted via a link on CST2555MyLearning website.
- The coursework has two parts:
  1. Report
  2. Tarball file
- The coursework has two problem definitions that are required to be solved with bash scripts
- The coursework is to be completed as an individual.
- All parts of the coursework have the same deadline indicated above.

1.3 Plagiarism

This coursework is to be completed as an individual and declared as your own work. Any work that indicates otherwise will be sent to the academic integrity panel for further investigation and follow University regulations.

Chapter 2

Create Files (8 marks)

2.1 Introduction

Essentially, the problem definition is similar to 1000Files available from the github resource used in exercises, which can be cloned with the following command:
`git clone https://github.com/iangmitchell/thousandFiles.git`.
However, there are some additional problems and improvements and it has to be written in bash script.

2.2 Problem Definition

Each filename has the following *three* character format, *xyz.txt*, as shown in Eq. 2.1.

$$\begin{aligned} &where \\ m \leq x, y, z \leq n \\ &and \\ m, n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \end{aligned} \tag{2.1}$$

The content of the file should be the same as the filename, e.g., *123.txt* should have content *123*. Every combination of the filename should be present, with one more addition to the directory structure.

To help organise the files they are to be stored in a directory *CWFiles*, level 0. The files are to be stored in the following paths: *CWFiles/x/y/xyz.txt*. Fig. 2.1 shows that level 1 directory matches x, and level 2 directories match y and stores all files beginning with xyz. So, `431.txt` would be stored in level 1 directory `4` and level 2 directory `3` as shown in fig. 2.1.

2.3 Example

The command *./createFiles 1 4* will produce a directory and file listing as shown in fig. 2.1

2.4 Verification

The bash script should check and display appropriate error messages for:

```
.
|-- 1
|   |-- 1
|   |   |-- 111.txt
|   |   |-- 112.txt
|   |   |-- 113.txt
|   |   `-- 114.txt
|   |-- 2
|   |   |-- 121.txt
|   |   |-- 122.txt
|   |   |-- 123.txt
|   |   `-- 124.txt
|   |-- 3
|   |   |-- 131.txt
|   |   |-- 132.txt
|   |   |-- 133.txt
|   |   `-- 134.txt
|   `-- 4
|       |-- 141.txt
|       |-- 142.txt
|       |-- 143.txt
|       `-- 144.txt
|-- 2
|   |-- 1
|   |   |-- 211.txt
|   |   |-- 212.txt
|   |   |-- 213.txt
|   |   `-- 214.txt
|   |-- 2
|   |   |-- 221.txt
|   |   |-- 222.txt
|   |   |-- 223.txt
|   |   `-- 224.txt
|   |-- 3
|   |   |-- 231.txt
|   |   |-- 232.txt
|   |   |-- 233.txt
|   |   `-- 234.txt
|   `-- 4
|       |-- 241.txt
|       |-- 242.txt
|       |-- 243.txt
|       `-- 244.txt
|-- 3
|   |-- 1
|   |   |-- 311.txt
|   |   |-- 312.txt
|   |   |-- 313.txt
|   |   `-- 314.txt
|   |-- 2
|   |   |-- 321.txt
|   |   |-- 322.txt
|   |   |-- 323.txt
|   |   `-- 324.txt
|   |-- 3
|   |   |-- 331.txt
|   |   |-- 332.txt
|   |   |-- 333.txt
|   |   `-- 334.txt
|   `-- 4
|       |-- 341.txt
|       |-- 342.txt
|       |-- 343.txt
|       `-- 344.txt
`-- 4
    |-- 1
    |   |-- 411.txt
    |   |-- 412.txt
    |   |-- 413.txt
    |   `-- 414.txt
    |-- 2
    |   |-- 421.txt
    |   |-- 422.txt
    |   |-- 423.txt
    |   `-- 424.txt
    |-- 3
    |   |-- 431.txt
    |   |-- 432.txt
    |   |-- 433.txt
    |   `-- 434.txt
    `-- 4
        |-- 441.txt
        |-- 442.txt
        |-- 443.txt
        `-- 444.txt

20 directories, 64 files
```

Figure 2.1: File and Directory output for `./createFiles 1 4` input

```
.
|-- createFiles.sh
|-- createFiles.sh~
|-- readFiles.sh
|-- readFiles.sh~
|-- test.sh
|-- CWFiles
    |-- 2
    |   |-- 2
    |   |   |-- 222.txt
    |   |   |-- 223.txt
    |   |   |-- 224.txt
    |   |   `-- 225.txt
    |   |-- 3
    |   |   |-- 232.txt
    |   |   |-- 233.txt
    |   |   |-- 234.txt
    |   |   `-- 235.txt
    |   |-- 4
    |   |   |-- 242.txt
    |   |   |-- 243.txt
    |   |   |-- 244.txt
    |   |   `-- 245.txt
    |   `-- 5
    |       |-- 252.txt
    |       |-- 253.txt
    |       |-- 254.txt
    |       `-- 255.txt
    |-- 3
    |   |-- 2
    |   |   |-- 322.txt
    |   |   |-- 323.txt
    |   |   |-- 324.txt
    |   |   `-- 325.txt
    |   |-- 3
    |   |   |-- 332.txt
    |   |   |-- 333.txt
    |   |   |-- 334.txt
    |   |   `-- 335.txt
    |   |-- 4
    |   |   |-- 342.txt
    |   |   |-- 343.txt
    |   |   |-- 344.txt
    |   |   `-- 345.txt
    |   `-- 5
    |       |-- 352.txt
    |       |-- 353.txt
    |       |-- 354.txt
    |       `-- 355.txt
    |-- 4
    |   |-- 2
    |   |   |-- 422.txt
    |   |   |-- 423.txt
    |   |   |-- 424.txt
    |   |   `-- 425.txt
    |   |-- 3
    |   |   |-- 432.txt
    |   |   |-- 433.txt
    |   |   |-- 434.txt
    |   |   `-- 435.txt
    |   |-- 4
    |   |   |-- 442.txt
    |   |   |-- 443.txt
    |   |   |-- 444.txt
    |   |   `-- 445.txt
    |   `-- 5
    |       |-- 452.txt
    |       |-- 453.txt
    |       |-- 454.txt
    |       `-- 455.txt
    `-- 5
        |-- 2
        |   |-- 522.txt
        |   |-- 523.txt
        |   |-- 524.txt
        |   `-- 525.txt
        |-- 3
        |   |-- 532.txt
        |   |-- 533.txt
        |   |-- 534.txt
        |   `-- 535.txt
        |-- 4
        |   |-- 542.txt
        |   |-- 543.txt
        |   |-- 544.txt
        |   `-- 545.txt
        `-- 5
            |-- 552.txt
            |-- 553.txt
            |-- 554.txt
            `-- 555.txt
```

Figure 2.2: File and Directory structure for `./createFiles 2 5` input. Notice the top level files, `createFiles.sh, readFiles.sh & test.sh` and their respective backups.

- Incorrect number of parameters entered
- Incorrect parameters entered, e.g., $m > n$
- Any file system errors concerning directories or files accessibility
- Any other exit status that is foreseen during the development of the bash script
- Execute the `test.sh` with appropriate error messages, as explained in ch. 4.

Chapter 3

Read Files (10 marks)

3.1 Introduction

The aim of this part of the coursework is to read the files created in the first. Then take each number in each file, add them to a total and finally, display the total. As with the first part the file must be written in bash script compatible with the version 4.4.20.

3.2 Problem Definition

The bash script has a filename `readFiles.sh`. It should take two parameters m and n.

Each filename has the following three character format as described in Eq. 2.1. The files are created using the bash script to solve the problem in Ch. 2, and so this is a progressive coursework, i.e., the second part can only be completed on successful completion of the first part, and is not unusual in bash script development. This means that the development of this part of the coursework cannot be completed until the development of the first part is complete, in other words the second part, `readFiles`, is dependent on the first part, `createFiles`.

So, given two parameters m and n, `./readFiles.sh` m n returns the sum of all files $xyz.txt$ governed by Eq. 2.1. There is a top level directory `CWFiles` that all subsequent directories are written to, the directory structure is the same as explained in Ch. 2 and illustrated in figs 2.1 & 2.2.

3.2.1 Verification

It is possible that the two parameters entered for the m and n for `./readFiles.sh` m n have different values than the original parameters entered for `./createFiles.sh` m n. The script development should take this into account and provide appropriate error messages for incorrect input.

Again, just like with `createFiles.sh`, execute the `test.sh` with appropriate error messages, as explained in ch. ch:eval.

This brings us to the next chapter on evaluation.

Chapter 4

Evaluation

Fig. 4.1 lists the test code to complete your evaluation, this is available to download from CST2555myLearning webpage.

Before final submission, `createFiles.sh` and `readFiles.sh` can be evaluated by running the test file. This should produce appropriate error messages and exit points from the bash script.

From the rubric 8 each successful pass of the test is awarded 1 mark.

```
#!/bin/bash
# test file for CW CST2555
#
#
tar
echo "============================="
echo "|  Start Create File Test   |"
echo "============================="
# Test 1
./createFiles.sh
./createFiles.sh 2 8
# Test 2
./createFiles.sh 2 4
# Test 3
./createFiles.sh 9 3
# Test 4
./createFiles.sh 0 9
echo "============================="
echo "| Finished Create File Test |"
echo "============================="
echo
echo "============================="
echo "|    Start Read File Test   |"
echo "============================="
# Test 5
./readFiles.sh
# Test 6
./readFiles.sh 9 3
./readFiles.sh 2 8
# Test 7
./createFiles.sh 3 7
./readFiles.sh 2 4
./readFiles.sh 3 7
# Test 8
./readFiles.sh 0 9
./createFiles.sh 0 9
./readFiles.sh 0 9
echo "============================="
echo "|  Finished Read File Test  |"
echo "============================="
```

Figure 4.1: Test file, `test.sh`, to run and evaluate coursework

Chapter 5

Documentation (2 marks)

Writing reports is good practice for any domain of Computer Science. Many institutions and organisations will have a template that you will have to comply to when writing documents. CST2555is no exception and an example of a coursework report is made available for you.

5.1 Comments & Indentation

Professional programmers will include appropriate comments and indentation in their code. The code submitted for this coursework is no exception and must include appropriate comments and indentation.

5.1.1 Comments

The problem with students writing comments is that until this point, you probably have never had to work with another team member and therefore have not had any feedback about your comments. You are the judge of your comments and because you write the code over relatively short periods of time, you seem to think the comments are obsolete or irrelevant.

Students also try to emulate comments that exist in textbooks; these are for the benefit of learners and can be very annoying to professional programmers. So, there is a fine balance between being annoying and commenting every line, to being abstract and not identifying intentions.

Please refer to the lecture on comments. This should help you, but in general bear in mind the following advice:

- Comments are to document intentions of the code, especially when the intention of the code is not obvious
- Comments are for other professionals
- Comments are not works of art - leave the ascii art for other projects
- Header comments for each file
- Comments are not version/source control - leave this to version control s/ware
- Low comment to code ratio

- Comments are for clarification and code intentions
- Comments should never be vindictive and void of emotion

5.1.2 Code Beautification

All code submitted is to be indented, each new structure is to be indented once to the right and at the end of that structure all code is to be indented left.

5.2 Report

The template of the report is made available in the example in MyLearning. Essentially, the report should have *four* chapters:

- Create Files Listing (with line numbers).
- Read Files Listing (with line numbers).
- This should follow the structure in the coursework example.
- References and citations are IEEE and alphabetically sorted by author, then by date (as used in this document).
- Written in English and third person (no I's or We's)
- All tables, listings and figures to have captions and uniquely numbered.

5.3 Anonymity

To avoid any subconscious biased when providing summative feedback anonymous marking is deployed on this module. This means that no identity is to be revealed to the tutor and the instructions are not to include any means of identification in the submission.

Chapter 6

Tar Ball (2 marks)

6.1 Introduction

Submission of code is to be compressed in a file named, `cw2555.tar.gz`, and submitted as a separate file to CST2555myLearning page. This file is to be uploaded and work on the Virtual Machine provided for CST2555and will be executed on bash version 4.4.20. The code will be tested using file listing in fig. 4.1 and non-compiling code will be penalised (see rubric in Table 8)

There are two files to submit: the tar ball; and, the report. The tar ball should be as follows:

- Compressed using gzip
- Include instructions to extract and decompress in report
- Have correct directory structure as follows shown in fig. 6.1

```
cw/
|-- createFiles.sh
|-- CWFiles
|-- readFiles.sh
|-- README.md
|-- test.sh
```

Figure 6.1: Tar Archive Structure. Top level directory `cw`. Includes `createFiles.sh, readFiles.sh and test.sh` bash scripts. A `README.md` file, with any instructions. A directory, `CWFiles`, which includes the files generated by `createFiles.sh`

Chapter 7

Feedback and Assessment

7.1 Quality Assurance

UK Higher Educational Providers (HEPs) have internal quality mechanisms that rely on External Examiners. Middlesex University is no exception and has internal quality mechanisms that examines and moderates your coursework submissions. Briefly, the internal quality mechanisms at Middlesex are:

- Module Leader/Tutor marks the assessment according to the marking criteria.
- Internal Moderator looks at a sample of the assessment to ascertain if the marking criteria is applied consistently and fairly.
- External Examiner, an independent from another HEP or Industry, looks at the same sample as the Internal Moderator and writes a report.
- The Programme Leader responds to this report.

In addition to this procedure, and in line with Middlesex University policy, anonymous marking is implemented for the assessment of this coursework, whereby the academic does not know the identity of the author of the coursework until entering the grades.

7.2 Feedback

Feedback is defined as information received from another individual to enchance one's understanding or knowledge. Students often perceive feedback as a grade, with some useful comments to: i) justify the assessment criteria; and/or ii) enhance the understanding of the topic. This perception is only partially correct, feedback comes in many forms, some are listed below:

- Lectures: have lots of opportunities to give feedback to students but many start with reinforcement of some principles learned prior to the lecture. Naturally, students asked questions in lectures and if answers are provided this is yet another form of feedback.
- Seminars: Working in groups and receiving comment from your colleagues is also a form of feedback, since it leads to improvement and better understanding.

- Labs: for many students the completion of practical work leads to a better understanding of the theoretical underpinnings of the subject domain studied. Usually, in these practical demonstrations there is some interaction with the tutor. Again, this is a type of feedback to help you understand the subject domain better.

The main two types of feedback identified in Higher Education are Formative and Summative. You will have many opportunities for formative feedback throughout this module and only one opportunity for summative feedback on each coursework.

7.2.1 Formative Feedback

Formative feedback aims to improve learning and is usually ungraded. Formative feedback occurs throughout the academic year. As your understanding increases this is partly due to formative feedback. Where formative feedback can be particularly useful in improving learning and understanding is before a coursework deadline. For this coursework there are two weeks of formative feedback: learning week 11 (after Multiple choice test) & 12 .

7.2.2 Summative Feedback

Summative feedback is always graded and is to assess performance on a module. These can often appear unfair or confusing. To ensure fairness and consistency, an assessment rubric has been completed. Rubrics are generally considered to be helpful in assuring that the consistency of marking has been applied throughout all courseworks [1, 4, 5].

This course is no different and uses the sections above to complete a rubric in table 8.

Chapter 8

Rubric

Criteria	Sub-criteria	0	1	1	1	1	W	Σ
Create File (8%)	Files	Incorrect structure or non-executable	Correct use of if-elif-else	Correct deletion & generation of files	Correct use of file descriptors	Correct use of for loops	1	/4
	Output	No output	Correct Error messages	Correct use of rm command	Correct user input	Correct use of case	0.5	/2
	Comments & Indentation	No comments or indent	Correct indentation	Anonymous Code	Appropriate var names	Comments concise and succinct	0.5	/2
Read File (10%)	Files	Incorrect structure or non-executable	Correct use of for	Correct use of loop to read file	Correct use of file descriptors	Correct use of if-elif-else	1	/4
	Sum	No sum	Convert leading zeroes	Correct sum displayed	Use of equations correctly	Find max & min directories	1	/4
	Comments & Indentation	No comments or indent	Correct indentation	Anonymity	Appropriate var names	Comments concise and succinct	0.5	/2
Document (2%)	Files	Incorrect structure or reveals identity	Correct structure and follows template	Numbering: chapter, section, page, figures, listings and line numbers	Use of "IEEE sorted" references and citations	Written in third person and relatively few mistakes	0.5	/2
Tar (2%)	TarBall	No code submitted	Code submitted, untarred	Code submitted and incorrectly tarred	Correct TarBall. Incorrect or no instructions	Correct TarBall, compression and correct instructions	0.5	/2
Evaluation (8%)	Create Files 1-4	No testing	Test 1	Test 2	Test 3	Test 4	1	/4
	Read Files 5-8	No testing	Test 5	Test 6	Test 7	Test 8	1	/4

Table 8.1: Rubric for Summative Assessment of Coursework 1

Bibliography

[1] Heidi Goodrich Andrade. Using rubrics to promote thinking and learning. *Educational leadership*, 57(5):13–19, 2000.

[2] D. J. Barrett. *Linux Pocket Guide*. O'Reilly, 3rd edition, 2016.

[3] Richard Blum and Christine Bresnahan. *Linux Command Line and Shell Scripting Bible*. Wiley, 3 edition, 2015.

[4] Phillip Dawson. Assessment rubrics: towards clearer and more replicable design, research and practice. *Assessment & Evaluation in Higher Education*, 42(3):347–360, 2017.

[5] Ernesto Panadero and Anders Jonsson. The use of scoring rubrics for formative assessment purposes revisited: A review. *Educational research review*, 9:129–144, 2013.

[6] Chet Ramey and Brian Fox. *Bash Reference Manual*, 2020.

[7] Ellen Siever, Stephen Figgins, and Aaron Weber. *Linux in a Nutshell*. O'Reilly.