University of Reading
Department of Computer Science

Julian Kunkel, Oliver Barnwell

Exercise week 2
CS1PR / Autumn 2019
30 Marks Total
Due Date: 2019-10-14 09:00

---

1. Make sure to carefully plan your time for the whole sheet, the complete exercise should represent approximately 5 hours of independent study. Each mark attained should correspond to roughly 10 minutes of your time and includes the time to check the lecture notes. If you are struggling then reach out for help at `http://cs-reading.slack.com`. Join here.

2. Remember to store the files in your CS-Git repository and use `https://hps.vi4io.org/cs1pr-submit` to upload and verify your submission.

3. Follow our code of honor: do not copy and plagiarize, any contribution made by others must be referenced correctly. Further information was given in the first tutorial. Before you attempt to lookup a solution in the Internet (or with colleagues), try for at least 1/3rd of the time to find a solution by yourself.

4. The table of contents contains the normal tasks and the alternative HARDER tasks marked in red. Only attempt the hard tasks, if you feel confident or have extra time to spend.

Exercise Introduction

Before attempting the exercises in this document please ensure that you have read and understood the key topics covered in Tutorial 1.

Contents

Task 1: Algorithmic Thinking (6 Marks)

One of the primary skills that a programmer must develop is the ability to break abstract and complex tasks down into a set of small, well defined and easily reproducible steps.
Imagine an everyday task that we as humans find relatively simple compared to computers, such as making a cup of coffee or driving a car and consider how you might program a computer to reproduce the same behaviour.

1. Begin by creating a text file: `everyday_algorithm.txt`. Use it to write down the steps of your algorithm. Also provide a short (between 50-100 words) description of your algorithm along with the steps.

   Your algorithm should:

   - **Differ from the given example(s)**
   - **Follow the format given below**
   - **Have at least 5 steps**
   - **Have some selections (e.g., yes/no decisions, see below for example)**

If we were to continue with the idea of analysing how a cup of coffee might be made, the steps might be:

```
1) Do we have coffee?
  -> Yes: Continue to (Step 2)
  -> No: Purchase coffee and return to (1)
2) Find a cup from the shelf
3) Is cup clean?
  -> Yes: Continue to (4)
  -> No: Clean cup and return to (3)
4) Measure coffee
5) Place coffee in cup
6) Pour water into kettle
7) Is water boiling?
  -> Yes: Continue to (8)
  -> No: check kettle is on and return to (7)
8) ..... and so on .....
```

2. Use LibreOffice to draw a flow chart of your algorithm. Refer to the tutorial section on FlowCharts if you need hints if you get stuck.

   Your flow chart should be a direct representation of the algorithm you submit in the text file from the previous step.

Submission (directory: `2/algorithmic-thinking`)

`2/algorithmic-thinking/fb.txt`	The feedback file for this exercise.
`2/algorithmic-thinking/algorithm.txt`	Text file containing a brief description of your algorithm and the steps (50-100 words).
`2/algorithmic-thinking/algorithm.odg`	LibreOffice file containing the flow chart representation of your algorithm.
`2/algorithmic-thinking/algorithm.pdf`	PDF export from LibreOffice of your algorithm flow chart.

Task 2: Algorithm Analysis (3 Marks)

The analysis of existing algorithms is a key skill of programmers as it is the preliminary to modify/extend code but also to understand your own code that you had written before.
For the following algorithm, execute it's behavior for n=10 and attempt to infer it's purpose.

```
Input: natural number n > 1

A is an array of values indexed by 2 to n; values initially all set to 1

for i = 2, 3, 4, ..., not exceeding n:
 if A[i] == 1:
   for j = 2*i, 3*i, ..., not exceeding n:
     A[j] := 0

Output: all i such that A[i]==1
```

Submission (directory: `2/algorithm-analysis`)

`2/algorithm-analysis/fb.txt`	The feedback file for this exercise.
`2/algorithm-analysis/output-n10.txt`	Text file containing the output for n=10, i.e., the list of i that meets the output criteria. Example format: 1,2,3,4,5
`2/algorithm-analysis/algorithm.txt`	Text file containing the behavior of the algorithm (at least 20 words).

Task 3: Bash Exercises (9 Marks)

We don't expect for you to be writing long and complex bash scripts straight away. We also appreciate that moving to a CLI (Command Line Interface), while powerful, can be challenging especially when you are accustomed to interacting with primarily graphical interfaces on Operating Systems like Windows.

Below we have included several Bash commands of varying complexity that also cover more advanced concepts of the Bash. We want you to experiment with them, try running them and observe the result! Think about what you expect their output to be before you run them. Perhaps make your own changes to the commands. Use this as an opportunity to learn about the shell and increase your comfort in writing your own scripts.
We would like you to produce a text file providing a short descriptions of your findings, i.e. all you need to provide is a brief description of what the command does (if possible write what you expected) and a short explanation of how it works where appropriate.

Run the following command **BEFORE** beginning the exercise, this command will create a directory and change your current directory to it:
`$ mkdir -p $HOME/exercise1 && cd $HOME/exercise1`
You should complete the steps below in this directory.

1. `$ echo "Hello World"`
2. `$ echo Hello, World`
3. `$ echo Hello, world; Foo bar`
4. `$ echo Hello, world!`
5. `$ echo "line one";echo "line two"`
6. `$ echo "Hello, world > readme"`
7. `$ echo "Hello, world" > readme`
8. `$ cat readme`
9. `$ example="Hello, World"`
10. `$ echo $example`
11. `$ echo '$example'`
12. `$ echo "$example"`
13. `$ echo "Please enter your name."; read example`
14. `$ echo "Hello $example"`
15. `$ three=1+1+1;echo $three`
16. `$ bc`
17. `$ echo 1+1+1 | bc`
18. `$ let three=1+1+1;echo $three`
19. `$ echo date`
20. `$ cal`
21. `$ which cal`
22. `$ /bin/cal`
23. `$ $(which cal)`
24. `$ ‘which cal‘`
25. `$ echo "The date is $(date)"`
26. `$ seq 0 9`

27. `$ seq 0 9 | wc -l`
28. `$ seq 0 9 > sequence`
29. `$ wc -l < sequence`
30. `$ for I in $(seq 1 9) ; do echo $I ; done`
31. `$ (echo -n 0 ; for I in $(seq 1 9) ; do echo -n +$I ; done ; echo) | bc`

Hints

- Use the up/down arrows to browse the history of the shell to find recently used commands
- Use the left/right arrows (Pos1/End keys) to change the position of the text cursor
- Use the tab key to autocomplete commands when you're halfway through typing them - it saves time!
- The middle mouse key copies previously marked text into the shell
- Bash has certain special characters such as `;!<>` that can cause unexpected behaviour when included outside of quotes in a command.
- Use the command `$ help` and `$ man` to find out what a specific command does.

Submission (directory: `2/introductory-bash`)

`2/introductory-bash/fb.txt`	The feedback file for this exercise.
`2/introductory-bash/findings.txt`	An enumerated list that provides for each command a prose sentence what the commands did. Where your expectation differs from observed behavior, include a sentence describing what surprised you. (at least 200 words).

Further Reading

- Shell scripting (also available as PDF) `https://www.shellscript.sh/`
- Shebang Wikipedia: `https://en.wikipedia.org/wiki/Shebang_(Unix)`
- Commands: `https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners`
- Command line structure: `https://www2.cs.duke.edu/csl/docs/unix_course/intro-14.html`
- `https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard`

Task 3: Bash Research (9 Marks)

This is a more difficult **optional** task which can be done instead of Task 3

We would like for you to produce a short (about 1 page, at least 300 words) report on the core concepts behind bash and shown them in practice. Your report should give brief explanations and examples of the following concepts:

- Special characters and escaping
- Redirection
- Piping
- Variables
- Arithmetic
- Common Commands that you find useful `beyond our examples (ls, cd, mkdir, ...)`

You should include examples from the alternative (easy) task.
In your report you should consider circumstances in which easily avoidable errors could be encountered and subsequently avoided in your commands (for example, unescaped special characters such as `;!#&`)

Submission (directory: `2/introductory-bash`)

`2/introductory-bash/fb.txt`	The feedback file for this exercise.
`2/introductory-bash/report.txt`	The 300 words report describing the concepts relevant to Task 1.

Further Reading

- `https://learnxinyminutes.com/docs/bash/` - Usage examples of many common bash commands.
- `https://www.gnu.org/software/bash/manual/html_node/Quoting.html` - Nice guide to how quotes work in Bash
- `https://jvns.ca/blog/2017/03/26/bash-quirks/` - Article going over basics and some common quirks in the language which can trip up beginners.
- `https://www.geeksforgeeks.org/piping-in-unix-or-linux/` - Covers piping and its application.
- Advanced Bash-Scripting Guide: I/O Redirection: `https://www.tldp.org/LDP/abs/html/io-redirection.html`

Task 4: Thinking like a programmer: Extracting Data (6 Marks)

This task will allow you to test your understanding of using what you have learnt about the Bash shell in practice. Below we present a real-world example of a task you might be asked to achieve using the shell.
We will use the Bash shell to extract data about certain pets from a file containing information about pets available in a shop. Therefore, we will execute a sequence of commands in a pipeline.

Follow these steps to get started:

1. Create a file called `animals.txt` with the following contents:

```
Oscar,Cat,12
Steve,Dog,7
Fred,Snail,3
Troy,Dog,2
Larry,Ant,1
Polly,Cat,15
Erik,Cat,14
Frank,Dog,11
```

   Description of the data-set:

   Columns: `Pet Name, Pet Type, Pet Age`

   Rows: `A single row represents an individual animal in the shop.`

2. Create a file for each question called: **petshop_question_n.sh** where n is the question number you're answering and ensure that it is executable: `chmod +x petshop_question_1.sh`. The shell script must read the file animal.txt and produce the requested output.

Questions:

Please refer to the **Hints** section. We provide some very useful commands with usage examples which will help you to complete the following questions. You should use these in your answer.

1. Sort the list of animals **numerically** by **age** in **ascending** order.

   *Hint: research how to sort numerically using the **sort** command. More information in the Hints section*

2. Sort the list of animals **alphabetically** by **name** in **ascending** order.

   *Hint: research whether the **sort** command has an option which lets you reverse the sort direction*

3. Print the **name, type and age** of the **oldest** animal in the pet shop.

   *Hint: the **head** command might come in useful here to retrieve only one row*

4. Print the **name and type** of the **oldest** animal in the pet shop.

   *Hint: remember the **cut** command can be used to select specific columns for output after sorting*

5. Print the **name, type and age** of only **dogs** without changing their order.

   *Hint: the **grep** command may come in useful to filter down to just dogs.*

6. Find the **name and age** of only **cats** and sort by **age** in **ascending** order.

   *Hint: the **grep** command may come in useful to filter down to just cats.*

Submission (directory: `2/petshop`)

`2/petshop/fb.txt`	The feedback file for this exercise.
`2/petshop/petshop1.sh`	Your solution for question one.
`2/petshop/petshop2.sh`	Your solution for question two.
`2/petshop/petshop3.sh`	Your solution for question three.
`2/petshop/petshop4.sh`	Your solution for question four.
`2/petshop/petshop5.sh`	Your solution for question five.
`2/petshop/petshop6.sh`	Your solution for question six.

Hints

- The following commands could be useful: **cut** - the **cut** command allows you to extract fields from lines provided as input.

  It comes in useful for this task as it allows us to extract fields of our choosing.

  Below is an example of how it may be used to extract only the names and ages of the animals, leaving out the animal type. Use the pipe `|` to forward output to the input of another program.

  - Input Command:

```
$ cut -d "," -f 1,3 aimals.txt
```

  - **-d**: We specify the **field separator** as the comma character using the **-d** argument: `-d ","`
  - **-f**: We specify the **field(s)** to select using **-f** argument. We want just the 1st and 3rd fields (name and age) so we specify it like so: `-f 1,3`. Check the manual page for other usage examples for this and other arguments.
  - Output Result (we expect only the names and ages of the animals):

```
Oscar,12
Steve,7
Fred,3
Troy,2
Larry,1
Polly,15
Erik,14
Frank,11
```

- **sort** - the **sort** command allows you to sort by a specified field. For example, the fields you extracted using the **cut** command.

  Below is an example of how it may be used to sort each row by the animal age in an ascending order.

- Input Command:

```
$ cat animals.txt | sort -k3 -n -t ","
```

- **-k**: We specify the **field(s)** to select using the **-k** argument. We want to select the age column to sort by, so we specify the third field: `-k3`
- **-n**: We use this argument to tell **sort** that we want to sort **numerically**, by default it sorts alphabetically.
- **-t**: We specify the **field separator** as the comma character using the **-t** argument: `-t ","`
- Output Result(we expect each row to be sorted by the animal age in ascending order):

```
Larry,Ant,1
Troy,Dog,2
Fred,Snail,3
Steve,Dog,7
Frank,Dog,11
Oscar,Cat,12
Erik,Cat,14
Polly,Cat,15
```

- **head** - the **head** command allows you to show the first **n** lines of whatever is provided as input. Where **n** can be any positive number.

  By default it will output the first **10** lines of whatever is provided as input.

  Below is an example of how we'd use **head** in conjunction with **sort** to return the youngest animal in the pet shop.

  - Input Command:

```
$ cat animals.txt | sort -k3 -n -t "," | head -n1
```

  - **-n**: We specify the **lines** to return using the **-n** argument. We want to return the first (youngest) animal in the shop so we pass **1**.
  - Output Result(we expect to see the row corresponding to the youngest animal):

```
Larry,Ant,1
```

- **grep** - the **grep** command allows you search the provided input for a specific value.

  As output it returns the lines which match the value provided.

Task 5: Thinking like a programmer: Understanding Pseudo-Code (6 Marks)

This task will test your understanding of pseudo-code and your ability to translate it into working shell script.

Fibonnaci Sequence

The Fibonnaci sequence is a sequence of numbers where each number in the sequence is equal to the sum of the two numbers preceeding it. The first two entries in the sequence are the numbers **0 and 1**.
A more formal definition of the function that computes the n-th fibonacci number is included below:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$$

The first **10** entries in the fibonnaci sequence (leaving out the zero) are therefore:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Fibonnaci Sequence Pseudocode

Below is the pseudocode to print the first 100 entries in the Fibonnaci sequence.

```
n = 10
a = 0
b = 1
for i in 0,..., n:
  print a
  temp = (a + b)
  a = b
  b = temp
```

Task

Take the pseudocode above and produce a shell script that outputs the Fibonnaci sequence **separated by commas** up to fibonacci number **n**, provided as the first argument to the script. Additionally, generate a flowchart that documents the algorithm.

Submission (directory: `2/fibonacci`)

`2/fibonacci/fb.txt`	The feedback file for this exercise.
`2/fibonacci/fib.sh`	`Synopsis: ./fib.sh <n>` **Usage Example:** **Input:** `./fib.sh 10` **Output:** `0,1,1,2,3,5,8,13,21,34,55`
`2/fibonacci/fibonacci.odg`	LibreOffice file containing the flow chart representing the algorithm.

Hints

- The variable `$1` contains the relevant argument to the shell script. `$0` is the program name, and any other subsequent number contains additional variables (if submitted).