Exercise 4.1.

Write a shell script that takes 3 arguments and prints them in reverse

order. If -h is entered anywhere a short description should be printed as well.

Exercise 4.2. (optional)

Write a shell script that does the following when given a path as first arg:

• If the path is a file, print whether it is executable and print the file size

• If the path is a directory cd to it and list its content.

Exercise 4.3. (optional)

Write a script that takes two integer values as args, I and J.

The script should:

• create directories named 1, 2, . . . , I

• Use touch to put empty files named 1 till J in each of these directories

• Print an error if a negative value is provided for I or J

• If any of the files exist, the script should exit with an error.

• Provide help if one of the args is -h, then exit the script.

• If the third argument is a file, the script should copy this file to all locations instead

of creating empty files with touch.

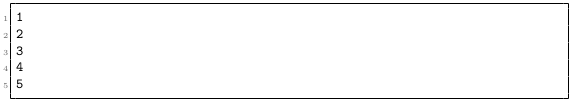
Exercise 4.4.

Implement the seq command in bash:

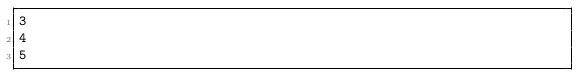
• If called with a single argument, print all integers from 1 to this value, i.e.



should give



• If called with two arguments, print from the first arg to the second arg, e.g. seq 3 5:



Assume that the first number is always going to be smaller or equal to the second

number.

• (optional) If called with three arguments, print from the first arg to the third in steps of the second, in other words

Gives:



Again assume that the first number is smaller or equal to the third one.

• Your script should print help if the first arguments is -h, and then exit.

• (optional) Your script should print an error if any of the assumptions is violated

and exit.

Exercise 4.6.

With this exercise we start a small project trying to recommend a book from Project Gutenberg based on keywords the user provides.

• Write a script that greps for a pattern (provided as an argument) in all books of

resources/gutenberg

– Make sure that your script keeps working properly if spaces in the pattern or

in the files are encountered

– Ignore case when grepping in the files

– You may assume all books of Project Gutenberg to end in the extension .txt.

– (optional) Provide help if the argument is -h

– (optional) Use proper error statements if something goes wrong or is not

sensible.

• Change your script such that it prints the number of matches and the number of

actual lines next to the script name. The fields of the table should be separated

by tabs (use echo -e).

A possible output could be



• (optional) Suppress the output of books without any match.

Exercise 4.7.

With your current knowledge of bash, propose two one liners that

• substitute all <tab> or <space> of a string in a variable VAR by <newline> char-

acters

• substitute all <newline> or <tab> characters by <space> characters

Hint: Both expressions have less than 30 characters

Exercise 4.8.

Write a script that takes the following arguments:

• -h, -q

• --help, --quiet

• -f followed by a filename

• anything else should cause an error message

Once the arguments are parsed the script should do the following

• Print help if -h or --help are present, then exit

• Check that the filename provided is a valid file, else throw an error and exit

• Print a nice welcome message, unless --quiet or -q are given

Exercise 4.9

Write a simple script read\_third.sh that outputs the third line provided on stdin to stdout and the fourth line to stderr. When you call it like



it should provide the output

and when called like

It should only print



Exercise 4.10.

• Write a script which asks the user for two numbers N and M (using read). and then

counts from N to M. You may assume that N << M.

• (optional) Lift the assumption and generalise your script such that it will count

from the smaller of N and M to the larger of N and M.

Exercise 4.11.

We want to write a more general version of exercise 3.2 in the previous exercise book

• Write a script takes the arguments --help, --from (followed by a line number)

and parses them. Deal with --help and detect unknown arguments.

• The default for --from should be the first line.

• Move the line of stdin given by --from to the last line on stdout, copy all other

lines

• You may assume that the users of your script are nice and only pass integer values

after --to or --from.

• If an error occurs, e.g. if the --to line number is larger than the number of lines

on stdin, inform the user.

• Now add an argument --to, which is followed by a number. It should have the

default setting of "end"(symbolising the last line on stdin)

• Assume (and check the input accordingly) that the value given to --to is larger

that the value to --from

• Change your code such that the line --from is moved to the line --to.

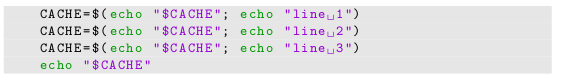
• Be careful when comparing line numbers to variables that may contain a string:



gives an error. This can be circumvented by guarding the [ with another [, e.g.

Exercise 4.12. (optional)

• Run the following lines of code



by pasting them into a script or just by running them in a terminal interactively.

Can you explain the result of the final echo?

• How would you alter the code in order to reverse the order in which the lines are

printed by echo. Try to achieve this without changing the order in which the

strings line 1, line 2 or line 3 appear.

• Use your above findings with the while read line paradigm to build a simple

bash version of the tac command, where all input on stdin is printed to stdout in

reverse order.

• Note: The final solution takes just 5 lines of bash.

Exercise 4.13

Recall that read can take more than one argument to read into multiple

variables at once.

• Assume you will get some data on stdin, which consists of a few columns separated

by one ore more <space> or <tab> characters. Write a script mtx\_third.sh that

prints the third column of everything you get on stdin.

• Try your script on some of the files in resources/matrices. E.g.



• How does it deal with multiple spaces compared to cut?

Exercise 4.14 (optional)

find is a really handy program to search for files and directories with uncountable options (see man find). You can find the most important options in the table below.

find per default searches through all directories and subdirectories

and prints the relative paths of all files satisfying the conditions to stdout. All options

you provide are connected using a logical and. This can of course all be changed (see

documentation).

If you have never used find before, try the following:

• find -name "\*.sh"

• find -type f -name "\*.sh"

• find $HOME -type d -name "\*bash\*"

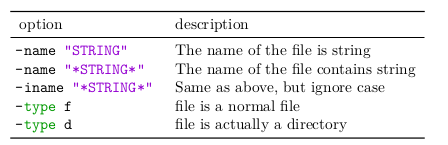
In this exercise you should build a grep\_all script:

• The script should search for all files in or below the working directory (using find)

• In all files found, the script should grep for the pattern provided on $1 and it

should print to stdout in which files and on which line the match occurred.

• The simplest way to achieve this is to pipe the output of find to while read line



Exercise 4.15.

The shell uses the following procedure to lookup the path of the com-

mands to be executed 7 :

• In a commandline the first word is always considered to be the command.

• If this word is a path (contains a “/”), execute this very file.

• Else go through all existing directories in the variable PATH. The directories are

separated using the character “:”. If there exists a file named like the command

in a directory, which is executable as well, execute this file.

• Else keep searching in the next directory in PATH

Example: The commandline



has the first word/command vim. Assume



Then a lookup reveals that the file /usr/bin/vim exists and is executable. So this file

is executed with testfile as the argument.

There exists a commandline tool, called which, that does exactly this lookup when

provided with a command as its first argument. See man which for more details. We

want to rebuild the which command as a script.

• Take the name of a command on $1

• Go through all existing directories in PATH and try to find an executable file called

$1 in these.

• If it exists print the full path and return 0

• Else return 1

Hints:

• Try to go through all directories in PATH first. There is an easy way to do this with

one of the loops we discussed and IFS-manipulation.

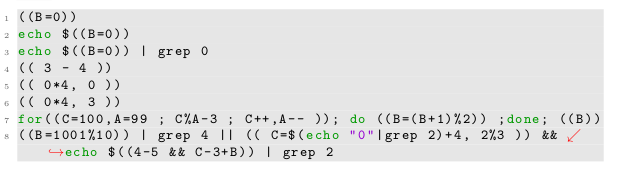
• Read the documentation of test in order to find ouf how to test if a file is exe-

Cutable.

Exercise 5.1

What is the return code of each line in the following code example and

why?



Exercise 5.2.

For the arithmetic expansion an empty variable or a string that cannot

be converted to an integer counts as zero (“C-false”)

• Try this in a shell or in a script, e.g. execute the following:



contrast this with

How could this behaviour (together with the [ program) be exploited to test

whether an input parameter can be properly converted to an integer?

• Write a script that calculates the cube of N, where N is an integer supplied as the

first argument to your script. Of cause you should check that N is a sensible integer

before entering the routine.

Exercise 5.3. (optional)

Use bash arithmetic expressions to calculate all primes between

1 and N, where N is a number supplied as the first argument to your script.

Exercise 5.4.

Now we want to extend our project to recommend books from Project

Gutenberg. Recall that your script from exercise 4.6 gives output of the form



where the columns were separated by tabs. The second column was the number of

matches and the third column was the number of actual lines in the file. Write a script

that

• takes one pattern as an argument, which is then used to call the script from exercise 4.6 e.g:



• parses the respective script output in the variable RESULT.

• calculates for each book the relative importance given as



and writes this ξ-value and the book name to a temporary file.

To make the next

steps easier you should separate the value and the book name by a <tab> and have

the ξ-value in the first and the book name in the second column.

• (optional) sorts the temporary file according to the relative importance

• (optional) suggests the 3 best-scoring books for the user and gives their score.

• (optional) One can entirely omit writing to a temporary file. Try this in your

script.

Try a few patterns, e.g. “Baker”, “wonder”, “the”, “virgin”, “Missouri, Kentucky”. Any

observations?

Exercise 5.5.

Write a script that takes either the argument -m or -s, followed by as

many numbers as the user wishes. The script should

• Calculate the sum of all numbers if -s is provided

• (optional) The mean if -m is provided

• (optional) Give an error if neither -m nor -s are given.

Some ideas:

• In both cases you will need to calculate the sum, so try to get that working first.

• As you know bc evaluates expressions given to it on stdin, so try to built an

appropriate sum expression from all commandline arguments using a loop. This

you echo to bc in order to get the sum.

• You may assume that users are nice and will only provide valid strings as the

number arguments to your script.

Exercise 5.6.

Implement the rev command in bash:

• Read input provided on stdin line by line.

• For each line reverse the characters, i.e.

Test → tset abcdef → fedcba

• Print the reversed string to stdout

Hints:

• The string reversal can be easily achieved using the substring expansion: By using

a length of 1 we can design an inner loop to extract one character after another

from the string.

• The new reverted string can than be built from these characters.