Exercises Part 1

Exercise 1.1.

Exploring the man program:

• Run the commands man -Lde tail and man -LC tail. What does the -L flag do to man?

• Find out about the different sections about the Unix manual (read line 21 till 41 of man man).

• Which section number is the most important for us?

• Find out how one can enforce that an article is only from a particular section.

Exercise 1.2.

A first look at Project Gutenberg books in resources/gutenberg

• Find out how many lines of the book pg74.txt actually contain “hunger”. Do this in two possible ways, both times using grep at least once.

– Once use at least one pipe

– Once use no pipe at all.

• Find out what the effect of the grep options -A, -B, -n, -H or -w is.

• (optional) pg74.txt contains two lines that directly follow another in which the

first line contains the word “hunger” and the second line contains the word “soon”.

Find out the line numbers of these two lines.

Exercise 1.3.

Looking at some matrices:

• Read the manpages of head and tail. Rebuild the effect of the tail com- mand using head. I.e. give a commandline that achieves the same effect as

< resources/digitfile tail, but that does not contain tail at all.

• Find out (using the manpage) how one could print all lines but the first of a file. You can either use the commands from your answer to the first part or use tail, both is

possible. Try your suggested command sequence on resources/matrices/3.mtx to see that it works.

• You might have noticed that the mtx files contain a few lines right at the top which begin with the special comment character “%”. Suggest a way to suppress comment lines in the file 3.mtx.

• Provide a sequence of commands using cut and sort which prints how many distinct values there are in the third column. I.e. if this column contains 3 fours, 2

threes and 1 zero, the answer should be 3. Note that the columns are not separated by tabs, so you will need to play with the flag -d of cut. Again use your idea

from the previous answer to ignore the comment line. You can check your result by looking at the file and comparing the output with your manual count.

• Provide a sequence of commands that prints the smallest value in the third column of 3.mtx. Again make sure your commands ignore the first comment line.

• Do the same thing with resources/matrices/bcsstm01.mtx. Be very careful and check the result properly. Here you will need the right options for sort to get the correct answer.

• Run the same sequence of commands as in the previous part on resources/matrices/lund\_b.mtx. The result should surprise you. What goes wrong here?

• Another tool that can be used to print certain columns in files is awk. The syntax is awk '{print $n}' to print the nth column. Use it instead of cut for the file

lund\_b.mtx. How does it perform?

Exercise 2.1.

Visualise the following command line as a redirection diagram

ls |& grep test | grep blub | awk '{ print $2 } ' & > outfile

Exercise 2.2

tee is a very handy tool if one wants to log the output of a long-running

command. We will explore it a little in this exercise.

• Imagine you run a program called some\_program which does a lengthy calculation.

You want to log all the output the program produces (on either stdout or stderr ) to a file log.full and all output that contains the keyword “error” to log.summary.

Someone proposes the commandline

some\_program | tee log . full |& grep error & > log . summary

Draw the redirection diagram. Does it work as intended? If not suggest an alternative that does achieve the desired goal making sure that only output from

some\_program actually reaches the log files.

• What happens if you run the command multiple times regarding the log files? Take a look at the manpage of tee and propose yet an alternative command line that

makes sure that no logging data is lost between subsequent runs of some\_program.

Exercise 2.3.

• Create a file called in and write some random text to it.

• Run < in cat > out. What happens?

• Run < in cat > in. What happens here?

• (optional) Run just plainly cat in a terminal. What do you observe? (Recall that you can quit any execution by Ctrl + C .)

Some hints to help you explore and explain what is going on:

– Draw a redirection diagram for just cat.

– Run cat followed by Ctrl + D . What happens?

– Read up on the keywords “end-of-file” or “EOF” in the bash manual and on wikipedia

Exercise 2.4.

To give you an idea why exit codes are useful as indicators what is going on, do the following

• Run a plain cat in your terminal:

$ cat

It hangs as expected after exercise 2.4 on the preceding page. Ctrl + D and check the return code by

Now type

$ echo $ ?

What is the output of the last command?

• Repeat the procedure using Ctrl + C instead of Ctrl + D . What is the result now? What is the reason for the difference, keeping the results of exercise 2.3 in mind.

Exercise 2.5.

Find out what the programs true and false do. Look at the following

expressions and try to determine the exit code without executing them. Then check yourself by running them on the shell. You can access the exit code of the most recent

command via echo $?.

false || true

true && false || true

false && false && true

false || true || false

Run the following commands on the shell

false | true

true | true

true | false

false | false

false |& true

What does the pipe do wrt. to the return code?

Exercise 2.6.

The main use of echo is to print all of its arguments to stdout. This is typically not needed a lot in interactive terminal sessions, but in fact one nevertheless

can make a lot use of echo to provide very particular input to another command using a pipe.

Keeping this in mind take a look at the following commands, which are all valid bash shell syntax. What do the commandlines mean? How are stdin, stdout and stderr of

grep connected to the shell environment? What is the exit code?

• echo test | grep test

• echo test & grep test

• echo test |& grep test

• echo test && grep test

• echo test || grep test

Exercise 2.7.

We already talked about the grep command in order to search for strings. One extremely handy feature of grep is that it returns 0 if it found a match and

1 otherwise. Change to the directory resources/gutenberg. Propose bash one-liners for each of the following problems.

• Print “success” if the file pg1661.txt contains the word “the” (there is a special grep flag for word matching), else it should print “error”.

• Do the same thing, but use a special flag of grep in order to suppress all output except the “success” or “error” in the end. Apart from there being less amount of

output, what is different?

• Now print “no matches” if pg1661.txt does not contain the word “Heidelberg”, else print the number of times the word is contained in the file.

• Try a few other words in the above command, like “Holmes”, “a”, “Baker”, “it”, “room” as well.

• Count the number of words in the file pg1661.txt

Exercise 2.8.

(optional) Go to the directory resources/directories.

• Run the rather confusing command

cd 3/3 || cd 4/2 && cd ../4 || cd ../3 && cat file

and explain what goes on in terms of the output printed on the terminal. Note, that this changes the working directory on the shell, so in order to run it

again, you need to cd back to resources/directories beforehand.

• Suggest the places at which we need to insert a 2>/dev/null in order to suppress the error messages from cd. Try to insert as little code as possible.

• Go back to the directory resources/directories. Now run

mkdir -p 3/3

to create the directory resources/directories/3/3. Explain the output of

cd 3/3 || cd 4/2 && cd ../4 || cd ../3 && pwd

As a general hint for this exercise: Try to run each command of the list in a shell and check the action as well as the return code each time, before moving on to the next

command which would run.

Exercise 2.9.

By crawling through the help provided by the help and the man commands, find out which of these commands are shell builtins: man kill time fg touch info history rm pwd ls exit

Exercise 3.1 (optional)

Write a bash script that produces its source code as output when executed. Hint: The solution has less then 20 characters.

Exercise 3.2.

This exercise is again considered with the matrices in resources/matrices.

• Write a script that copies all data from resources/matrices/3.mtx to output.mtx with the exception that the first (comment) line should appear at the very end of

the file output.mtx

• In other words the net effect should be that the script moves the comment line to the end of output.mtx

Now generalise the script: Make use of the positional parameters in order to:

• Write a script that takes two arguments: The first should be a matrix file, the second should be an output file, to which the script will write all data.

• The script should again copy all data over from the matrix file to the output file, with the exception that the comment line appears at the end of the output file.

Exercise 3.3. (optional)

Remind yourself that all commands in a script are connected to the script’s stdin and stdout.

(a) Write a script, which takes a keyword as first argument and greps for this keyword on all data supplied on stdin. Test it with a call like

(b) Adjust your script to only print the first matching line.

(c) Discard what you did in (b) and now print only the last matching line.

(d) Now try to combine (a), (b) and (c): The script should now print only the first and the last matching line, then an empty line (just a plain echo) and then all

matching lines including the first and the last, exactly as they are returned from the initial grep you used in (a). Most importantly the script should always print

all these things in exaclty the given order. You will most probably run into problems. Read on to get an idea how to solve them.

Achieving part (d) of the exercise is a bit tricky, since both the stdin and stdout of are pretty volatile.

Because they both are so-called streams everything which is received on stdin or sent to stdout is gone immediately and cannot be processed again.

### In order to be able to use for example stdin twice in the same script, one can make use of the following trick:

# Cache from stdin

CACHE = $ ( cat )

# Use it once

echo " $CACHE " | grep ...

# Use it twice

echo " $CACHE " | grep ...

here the double quote “"” are again neccessary to keep the line breaks.

(e) Try to understand how this works in light of what we discussed earlier

(f) Use this (or something similar) to finally solve part (d)

Exercise 3.4

Open the script 3\_simple\_scripts/ex\_quoting.sh

which is supposed to search for a keyword in a few selected Project Gutenberg books. Right now it does not quite work as expected.

Identify and correct possible problems.

Exercise 3.5

It is very common to see the paradigm

echo " $VAR " | wc -l

in order to count the number of lines in the variable VAR. Try this for the following

values of VAR:

• VAR=$(echo line1; echo line2), i.e. two lines of data

• VAR=$(echo line1), i.e. one line of data

• VAR="", i.e. no data at all

Can you describe the problem? There exists an alternative method to count the number

of lines, which is more reliable, namely

echo -n " $VAR " | grep -c ^

You will learn in the next lesson that the -n flag prevents echo from printing an extra

trailing <newline> character after the content of VAR has been printed.

The parameter ^ which is passed to grep is a so-called regular expression, which we will discuss in more

detail in chapter 7 on page 92. For now it is sufficient to know that ^ is a “special” kind of keyword that matches all beginnings of all lines.

• Try this command on the three examples above to verify that it works.

Exercise 3.6

(optional) Write a script that

• takes a pattern (which may contain spaces) as an argument.

• uses recursive ls (manpage) to find all directories below the current working directory, which have a relative path, that matches the pattern.

• prints the relative paths of these matching directories.

For example: If the current working directory contains the directory resources/matrices as well as the directory resources/gutenberg, and the pattern is “gut”, the script should

print resources/gutenberg but not the other path.

A few hints:

• First run ls --recursive once and try to understand the output

• What distinguishing feature do directory paths have compared to the other output printed?

• Everything can be achieved in a single line of bash using only 3 different programs (ls, grep and one more).

• You might need to make the assumption that none of the files or directories below the working directory contains a “:” character in their name in order to achieve

Exercise 3.7

In this exercise we want to write a script that searches for keywords in a file and displays how many findings there were and where these were found.

• Familiarise yourself with the way the -n flag changes the output of grep. How could you use this together with cut to find all line numbers where a particular

keyword was found?

• Proceed to write a script that takes a filename as first argument and a search word a second argument. Return the line numbers where the word was found.

• Now also display a summarising message, which shows how many matches were found.

• Test your results for some keywords and a few project gutenberg books.

• Now take a look at the exit command (help exit). It can be used to abort a script prematurely and provide a return code to the caller. Use it to amend your

script such that it returns 0 if any match is found and 1 otherwise.

• Count the number of characters of your script, excluding comments (use the script resources/charcount.sh for this task). The shortest shell script (using only what we have covered so far) wins :).