# Assignment 1: Unix command line

Autumn 2020

This first assignment will make extensive use of the Unix command line. Make sure you are at a computer that gives you access to one of these, like the Lab computers, a Mac or Linux machine, or Windows with WSL installed.

This assignment starts out as copy-and-try exercises, where you are asked to hand in a slight variations of what you have just been shown. You are strongly encouraged to type in all the examples in this assignment and try them and play around with them until you understand what is going on. Remember, you can use the up and down arrows to pull up previously used commands, and the left and right arrows if you want to move the cursor within the command you are currently editing.

The things to hand in are numbered. Write down the commands you use and a bit of prose explaining why you decided to do so.

Hand in the answers to the assignments in Canvas, either by uploading a text file or by writing your answers directly in the webform. Do not submit other file formats, for instance ZIP archives, PDFs, doc-files etc.

## Preparation

You will need to download a zip archive with materials. The zip file can be found together with this PDF on Canvas. Download it and move it to some good place to work with the exercises, making sure you can find it again from the command line.[1] You can unpack the zip archive using the unzip command:

```
$ unzip a1_materials.zip
```

The materials are in two subdirectories. Take a look around to see what is there. Use the `less` command to inspect some files. For instance, assuming you are in the correct subdirectory, try:

```
$ less chapter01.txt
```

As a reminder: pressing `q` gets you out of `less`, just like with the `man` pages.

[1]For those of you who have installed the Windows Subsystem for Linux, putting it somewhere in the `LW_share` directory would be the recommended place.

## 1 Searching through multiple files: finding and counting names

In the subdirectory `chapters` are the twelve chapters of *Alice's adventure in wonderland* in different files. As you know, the `ls` lists the contents of a directory, and you can use wildcards to view only a part of the contents. For instance, the following lists only chapters 10 and upwards:

```
    $ ls chapter1*.txt
```

**Q1** What would the `ls` command look like if you wanted to list all seperate chapters, but not the files with front- and backmatter, and not the file containing the combined chapters called `chapters_all.txt`?.

The command `grep` lets you look for words or patterns in text files. The following command gives you all mentions of the word *Dinah* in the first chapter of Alice's adventure:

```
    $ grep Dinah chapter01.txt
```

Now, instead of just searching in one file, we can search in multiple files, for instance using the wildcard syntax we just saw with `ls`.

**Q2** What does the command look like to find all instances of *Dinah* in any of the chapters, but not in the the front- and backmatter?

The tool `grep` can be run with an option to make it return the number of matches, rather than the matching lines itself. So, for instance, we have:

```
  $ grep -c Dinah chapter01.txt
  4
```

We can use this option, together with the filename wildcard you wrote before, to look for the number of occurrences of the word *Alice* in each of the chapters.

**Q3** What command would achieve this?

A list of how often the name *Alice* occurs in each chapter is much more interesting if we know the length of each chapter in words. The `wc` command can give you this information. Read the `man` page for `wc` to learn how you can get this information. When you find out how to do so, use the chapter sizes to answer the following questions:

**Q4a** which chapter has the most mentions of Alice relative to its size, and

**Q4b** how many occurrences of *Alice* per word does this chapter have?

Include the commands you used to find out and a description of how you calculated the answer. Note: it may involve doing some calculations yourself.

## 2 Combining commands in pipelines: Making a list with word counts

(The rest of this assignment is based on a Unix tutorial called ‘Unix for poets’ by Kenneth Church.)

In the other subdirectory `tokenized`, you find a specially prepared version of Lewis Carroll’s book. This version has been tokenized, which means that every *token* – everything that we wish to count as a word-like unit – is separated from its surrounding tokens. In this file, tokens are seperated by spaces. In addition, all capitalization has been removed, and each paragraph is on one line.

**Q5** How many paragraphs are in the book?

We are interested in getting a list of words, with for each word the number of times it occurs in the book. The underlying idea – or algorithm – we will be using is this:

1. first create a sorted list of tokens, so that all occurrences of the same token are adjacent,
2. then pass over this list, replacing all repeated occurrences of a token by just one and the number of replaced occurrences.

The first step can be handled by the `sort` command. If you have a look at its `man` page, however, you will see that `sort` sorts lines. We therefore have to start by putting each token on a seperate line.

We know that each token is seperated by a space. If we could replace each space by a newline – a line break – we would have one word per line. The `tr` command can do this for us:

```
$ tr ' ' '\n' <tokenized.txt | head -n5
chapter
i
.
down
the
```

In usage above, the `tr` command takes two arguments: a list of characters to replace and list of corresponding replacements. In our case, the first list only contains the space, and the second list only the newline.

**Q6** Change the command, so that the complete output of `tr` is written to a file `one_token_per_line`.

Make sure you inspect the file so that you’re confident it is the whole of Alice’s adventure, one token per line.

Let's put `sort` into the mix:

```
$ sort one_token_per_line | head -n5
!
!
!
!
!
```

This is not very exciting, but remember that this is exactly what we wanted: a list with repeated occurrences of a token on adjacent lines.

**Q7** Replace `head` by a call to `tail`. What do you see?

**Q8** Do we have to use the intermediate file `one_token_per_line`? What do you think are the pros and cons of using such an intermediate file?

The next step, according to the algorithm, would be to count repeated mentions. The command `uniq` helps us with that, like so:

```
$ sort one_token_per_line | uniq -c | head -n5
   450 !
    51 (
    56 )
    60 *
  2418 ,
```

It may be interesting to see this list not in alphabetical order, but sorted after frequency. Since the frequency is first on each line, we could use `sort` again to find the most frequent tokens:

```
$ sort one_token_per_line | uniq -c | sort -nr | head -n10
  2418 ,
  1638 the
  1115 ‘
  1112 ’
   989 .
   866 and
   725 to
   631 a
   595 it
   553 she
```

This second call to `sort` has two options set: `-n` and `-r`. Take a look at the manual for `sort` again, and play around by omitting one or both of these options.

**Q9** Try to describe why we need those options?

## 3 Counting pairs of words

Language is not just about single words in isolation. We can also ask ourselves what token combinations often occur. If we could get two subsequent tokens per line instead of one, we could use the same `sort-uniq-sort` technique as above. The `paste` command helps us here. It allows you to combine text files as if they were columns.

```
$ paste one_token_per_line one_token_per_line | head -n5
chapter chapter
i i
. .
down down
the the
```

The problem here should be obvious: by pasting the file to itself, we just get each token repeated. What we would like is for the second column to be offset by one line. The `tail` command can make one of those offset files for us, like so:

```
$ tail -n+2 one_token_per_line | head -n5
i
.
down
the
rabbit-hole
```

**Q10** Compare this to the output of `head -n5 one_token_per_line`, and make sure you understand what you are seeing here.

Now, assume you directed the output of `tail` to a file `one_token_per_line_o1`. Pasting the the two files then gives:

```
$ paste one_token_per_line one_token_per_line_o1 | head -n5
chapter i
i .
. down
down the
the rabbit-hole
```

Finally, let's put the whole thing together (the > on the second line is a prompt, not part of the command):

```
$ paste one_token_per_line one_token_per_line_o1 | sort | uniq -c |
> sort -nr | head -n5
   460 ,and
   427 . ‘
   397 ,’
   330 ’ said
   282 ! ’
```

It is unfortunate, but also expected, that this top 5 list mainly contains punctuation. Perhaps the pairs just below the most frequent ones are more interesting?

**Q11** Use a combination of `tail` and `head` to find numbers 11–20 in the list of most frequent word pairs.

## 4 Counting word triples

This final section only consists of assignments:

**Q12** Extend the method for counting pairs to counting three-token combinations. What are the 20 most common combinations?

The problem of punctuation marks dominating the most frequent combinations has become even worse. So let us filter these combinations out. Using `grep -v` puts `grep` in reverse: instead of only showing matching lines, it shows non-matching lines.

**Q13a** If you know / manage to figure out how to use regular expressions in `grep`: use that; otherwise: stack different `grep` filters in a pipeline to exclude the offending combinations.

**Q13b** What are the 20 most common combinations?